Universidad
Carlos III de Madrid

Ingeniería Técnica de Telecomunicación.

Especialidad Sonido e Imagen.

PROYECTO FIN DE CARRERA

Desarrollo de una interfaz *web* basada en Flash para una aplicación de reconocimiento de imagen

Autor: Diego Serrano Toca.

Tutor: D. Jesús Arias Fisteus.

Leganés, diciembre de 2011

Título: Desarrollo de una interfaz *web* basada en Flash para una aplicación de reconocimiento de imagen.

Autor: Diego Serrano Toca.

Director: D. Jesús Arias Fisteus.

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO **PRESIDENTE**

Resumen del Proyecto.

El Proyecto realizado consiste en el desarrollo de una aplicación que permite al usuario corregir exámenes de tipo test mediante el uso de una cámara *web*.

En primer lugar, el usuario debe enviar al servidor un archivo con la configuración del examen. Si lo desea, también puede enviar la lista de alumnos para relacionar cada examen con un NIA (Número de Identificación de Alumno).

Por otro lado, antes de empezar con las capturas, el usuario tiene la posibilidad de elegir entre comenzar la corrección de un nuevo examen o continuar corrigiendo uno ya existente en su disco duro.

Una vez enviados los archivos de configuración el programa remite las capturas de la cámara *web* al servidor hasta que éste decida que hay una imagen adecuada para ser corregida. Una vez procesada, el servidor le devuelve al programa la corrección del examen. Ésta se muestra sobre la imagen del examen y es plenamente modificable por el usuario.

Para terminar, una vez que el usuario ha dado la corrección por buena, éste puede elegir entre corregir otro examen o por el contrario cerrar la sesión. Cuando el usuario finalice la corrección de todos los exámenes que necesite, el programa le da la opción de guardar dichos datos en su disco local antes de salir definitivamente del programa.

Al finalizar la sesión (una vez guardados o no los datos), el programa nos lleva de nuevo a la pantalla inicial donde el sujeto podría volver a empezar con todo el proceso.

Abstract.

This project undertakes the development of an application which enables users to correct multiple-choice exams by means of a webcam.

First of all, users must send a file to the server with the exam configuration. Optionally, users can also send the list of participating students to link each exam to a SIN (Student Identification Number).

Additionally, before commencing with the image capturing, users have the option of correcting a new exam or continuing the correction of an existing exam, already loaded in the hard drive.

Once the configuration files have been loaded the program sends the images captured by the webcam to the server until it determines that there is an image which can be corrected. Once processed, the server sends the exam correction back to the program, which appears super-imposed on the exam image and may be modified completely be the user.

Finally, once the user has validated a correction he can choose between correcting another exam or closing the session. When the user has finished correcting all the exams he/she needs, the program gives him/her the option of saving the data on his/her local drive before finally closing.

When a session is closed (having saved the data or not) the program takes the user back to the start-up screen, from which he can begin the process again.

Índice de contenidos.

Capítulo 1.- Introducción.

1.1.- Motivación:

La motivación para el desarrollo de este Proyecto fin de carrera es la necesidad de implementar una aplicación para la corrección de exámenes mediante el uso de una cámara *web*. Dicha aplicación debe poderse ejecutar en un navegador *web* y acceder a la herramienta E*yegrade*. La herramienta sirve para la corrección de exámenes de tipo test. La aplicación se ejecuta de forma remota sin necesidad de instalar ni la misma ni *Eyegrade* en el equipo de trabajo del usuario.

Además de darle al usuario la posibilidad de ir almacenando dichas correcciones en su disco local para futuras consultas y modificaciones sobre las mismas.

La herramienta *Eyegrade*, desarrollada por el profesor D. Jesús Arias Fisteus, permite la corrección de exámenes de tipo test a partir de imágenes de la cuadricula donde los alumnos marcan las respuestas a las preguntas del examen. Al tener ya disponible la parte que se va a encargar de la corrección de la imagen, en el Proyecto habrá que desarrollar la parte que comunique el equipo del usuario (básicamente la cámara *web*) con el servidor donde está alojada la herramienta para que el usuario pueda corregir sus exámenes.

1.2.- Objetivos:

Los objetivos marcados para la completa realización del Proyecto son:

- ✓ Acceder a las cámaras web conectadas al equipo del usuario.
- ✓ Realizar capturas de imágenes mediante la cámara *web* seleccionada por el usuario.
- ✓ Enviar dichas imágenes al servidor para su posterior corrección.
- ✓ Reducir al máximo el ancho de banda que se consume en el envío de dichas imágenes para hacer la comunicación más ágil.

- ✓ Recibir la respuesta del servidor y colocar los símbolos de la corrección sobre la imagen del examen.
- ✓ Almacenar los resultados cuando el usuario decida que no quiere seguir corrigiendo más exámenes.

1.3.- Plan de trabajo:

Diagrama de Gantt incluido en el apéndice 5, además de una tabla con los nombres de las tareas y las fechas de inicio y fin de las mismas.

1.4.- Contenido de la memoria:

El capítulo 2 contiene todo lo referente a las distintas tecnologías estudiadas para llevar a cabo la resolución de este Proyecto de fin de carrera. Tanto las finalmente empleadas como las que por uno u otro motivo fueron finalmente descartadas.

El capítulo 3 describe todos los requisitos previos que debe cumplir la aplicación una vez esté finalizada.

El capítulo 4 describe cuales son las entidades implicadas en el Proyecto. En este capítulo se analiza el mismo diferenciando entre la parte de la que se encarga el cliente y la del servidor. También explica cómo es la comunicación entre ambas entidades. Así mismo detalla las diferentes pantallas que se va a ir encontrando el usuario a lo largo de la ejecución de la aplicación. También se explica lo que hace el programa cuando el usuario presiona alguno de los botones disponibles en la misma.

El capítulo 5 describe en profundidad las partes más significativas del código desarrollado para el correcto funcionamiento del programa de una forma descriptiva. No incluye partes del código ya que éstas vienen completas en el anexo.

El capítulo 6 explica las diferentes pruebas que se han ido realizando a la par que se iban desarrollando nuevos módulos para poder ir acoplando unos con otros.

El capítulo 7 da unas conclusiones finales sobre el Proyecto así como una serie de opciones por las que este Proyecto podría ser explotado en otros ámbitos distintos al de la corrección de exámenes de tipo test.

Capítulo 2.- Estado del arte.

En este capítulo se van a comentar las diferentes tecnologías disponibles para la realización de este Proyecto. Se centra la atención en *Flash* y *Eyegrade*, que son las que finalmente se han utilizado.

2.1.- *Flash*.

Flash es una aplicación de creación y manipulación de gráficos vectoriales. Dispone del lenguaje *ActionScript* para manejar el código y utiliza elementos multimedia (audio y video), para producir contenido interactivo. Éste se ordena en una línea temporal mediante fotogramas.

Flash es un entorno de desarrollo y *Flash Player* es el reproductor usado para visualizar las aplicaciones generadas. El reproductor se distribuye de forma gratuita y está ampliamente extendido. Los contenidos generados pueden encapsularse en una página *web* o ser reproducidos independientemente con *Flash Player*.

Según [2]:

<<*ActionScript* es un lenguaje orientado a objetos que permite ampliar las funcionalidades que *Flash* ofrece en sus paneles de diseño y además permite la creación de películas o animaciones con altísimo contenido interactivo. Provee a *Flash* de un lenguaje que permite al desarrollador añadir nuevos efectos o incluso construir la interfaz de usuario de una aplicación compleja. La versión 3.0 de *ActionScript* ha marcado un cambio significativo en este lenguaje, puesto que se ha encaminado a ser un lenguaje orientado a objetos solamente a través de clases.>>

Para la realización de este Proyecto se ha elegido utilizar esta versión de *ActionScript*, cuyas principales características, según [3], se van a detallar a continuación:

- ✓ <<*ActionScript* 3.0 ofrece un modelo de programación robusto que resultará familiar a los desarrolladores con conocimientos básicos sobre programación orientada a objetos.>>

- ✓ <<Una nueva máquina virtual *ActionScript*, denominada AVM2, que utiliza un nuevo conjunto de instrucciones de código de *bytes* y proporciona importantes mejoras de rendimiento.>>

- ✓ <<Una base de código de compilador más moderna, que se ajusta mejor al estándar ECMAScript (ECMA 262) y que realiza mejores optimizaciones que las versiones anteriores del compilador.>>

- ✓ <<Una interfaz de programación de aplicaciones (API) ampliada y mejorada, con un control de bajo nivel de los objetos y un auténtico modelo orientado a objetos.>>

- ✓ <<Un núcleo del lenguaje basado en el próximo borrador de especificación del lenguaje ECMAScript (ECMA-262) edición 4.>>

- ✓ <<Una API XML basada en la especificación de *ECMAScript* para XML (E4X) (ECMA-357 edición 2). E4X es una extensión del lenguaje *ECMAScript* que añade XML como un tipo de datos nativo del lenguaje.>>

- ✓ <<Un modelo de eventos basado en la especificación de eventos DOM (modelo de objetos de documento) de nivel 3.>>

El código *ActionScript* 3.0 puede ejecutarse con una velocidad diez veces mayor que el código *ActionScript* heredado. Aumenta las posibilidades de creación de *scripts* de las versiones anteriores.

En cuanto al API cabe destacar la gran cantidad de ejemplos de los que dispone para que sea más sencillo al desarrollador empezar a utilizar diferentes controles.

Dentro de las clases utilizadas para la realización de este Proyecto cabría destacar las siguientes, según [4]:

- ✓ <<La clase *URLRequest* captura toda la información en una sola solicitud HTTP.>>

- ✓ <<La clase *URLLoader* proporciona un mecanismo independiente para cargar texto y datos binarios en aplicaciones basadas en datos.>>

- ✓ <<La nueva clase *Loader* proporciona una forma de acceso a información detallada sobre el contenido cargado.>>

- ✓ <<La clase *ByteArray* permite optimizar la lectura, escritura y utilización de datos binarios.>>

- ✓ <<La clase *BitmapData* le permite trabajar con los datos (*pixeles*) de un objeto *Bitmap*.>>

- ✓ <<La clase *Camera* permite capturar vídeo de una cámara conectada a un equipo que ejecute *Flash Player*.>>

- ✓ <<La clase *Event* se utiliza como clase base para la creación de objetos de eventos, que se transmiten como parámetros a los detectores de eventos cuando se produce uno.>>

- ✓ <<La clase *FileReference* proporciona un medio para cargar y descargar archivos entre el equipo de un usuario y un servidor. El sistema operativo pide al usuario mediante un cuadro de diálogo que seleccione un archivo para cargar o una ubicación para descargar.>>

- ✓ <<La clase *Graphics* contiene un conjunto de métodos que puede utilizar para crear una figura vectorial.>>

- ✓ <<La clase Video muestra vídeo grabado o en vivo en una aplicación sin incorporar el vídeo al archivo SWF. Esta clase crea un objeto Video en una interfaz de *Flash* que reproduce cualquiera de los siguientes tipos de vídeo: archivos *Flash* Video (FLV) grabados y almacenados en un servidor o localmente, o el vídeo en vivo capturado del equipo de un usuario.>> Éste ultimo tipo es el que interesa para la realización de este Proyecto.

- ✓ <<La clase XML contiene métodos y propiedades para trabajar con objetos XML. La clase XML implementa potentes normas de gestión de XML definidas en la especificación *ECMAScript* para XML (E4X) (ECMA-357 edición 2).>>

2.2.- Tecnologías finalmente descartadas.

Dentro de este apartado caben destacar dos tecnologías fundamentalmente, como son *Silverlight* y HTML5.

2.2.1.- *Silverlight.*

Según [5]:

Silverlight es un complemento desarrollado por *Microsoft* para cubrir las mismas necesidades que hasta ese momento cubría *Flash*. Se distribuye de forma gratuita y añade a los navegadores la capacidad de reproducción multimedia.

Según [6]:

Como no es compatible con sistemas basados en GNU/Linux, dispone de un complemento llamado *Moonlight* desarrollado por el Proyecto Mono en colaboración con *Microsoft*, que si es compatible con GNU/Linux.

Según [7]:

Silverlight permite el uso de los dispositivos de audio y video disponibles en el equipo del usuario. De un modo sencillo se puede implementar una aplicación que se encargue de mostrar la captura, así como una lista con todos los dispositivos instalados.

Según [8]:

La comunicación entre la aplicación *web* y el servidor es un poco compleja en *Silverlight*. Para empezar solo permite la comunicación mediante sockets en los puertos comprendidos entre el 4502 y el 4534, si se intenta conectar con otro puerto desde una aplicación implementada en *Silverlight* dicha petición será denegada automáticamente. Además solo soporta comunicaciones mediante el protocolo TCP.

2.2.2.- HTML5.

Según [9]:

HTML5 es la quinta versión del lenguaje HTML. Está siendo desarrollado en paralelo con XHTML y aún se encuentra en modo experimental.

Para poder utilizar este lenguaje es necesario que los usuarios actualicen sus versiones de navegadores.

Ésta es la primera versión de HTML que incluye elementos que proporcionan funcionalidades multimedia.

Según [10]:

HTML 5 aún no implementa la funcionalidades necesarias para el manejo de la cámara *web* que tenga el usuario instalada en su equipo, lo que descarta esta tecnología para ser empleada en este Proyecto.

Según [11]:

HTML 5 dispone de un elemento *Canvas* en el que es realmente sencillo pintar símbolos mediante el uso de *JavaScript*. Esto es importante ya que es necesario poder pintar los símbolos de la corrección enviada por el servidor.

Según [12]:

HTML 5 también permite otro de los requisitos previos al inicio del Proyecto, como es el envío de imágenes al servidor. Haciendo uso de *JavaScript* se envían los datos contenidos en el elemento *Canvas* al servidor.

Por desgracia como aún no se pueden realizar las capturas desde la cámara del usuario no importa que se pueda enviar la información si no se tiene disponible.

2.2.3.- Conclusiones.

En esta parte se van a exponer los motivos por los cuales se ha elegido *Flash*, frente a *Silverlight* y HTML5, para la realización de este Proyecto.

Una vez analizadas las posibilidades que *Flash* brinda a los desarrolladores y enfrentadas éstas con las necesidades para este Proyecto en concreto, se llega a la conclusión que *Flash* es un herramienta suficientemente potente para desarrollar la aplicación.

Los motivos por los que se la ha elegido finalmente para desarrollar la aplicación son:

- ✓ La gran implantación de *Flash Player* en los equipos de los usuarios finales de

la misma, ya que prácticamente todos los usuarios de internet disponen de alguna versión del *Flash Player* que además se distribuye de forma gratuita. Lo que implica que el usuario final de la aplicación desarrollada en este Proyecto no necesitará instalar ningún complemento adicional en su equipo.

- ✓ Otro aspecto importante es que *Flash* es multiplataforma, lo que permite ejecutar la aplicación en varios sistemas operativos, amén de en varios navegadores.

- ✓ También es importante destacar el poco tamaño que tienen los programas generados con *Flash* y la facilidad con la que pueden ser encapsulados dentro de una página *web*.

- ✓ Otra de las capacidades de *Flash* importantes para el desarrollo de ésta aplicación ha sido la posibilidad real de introducir código en *JavaScript* dentro del código de *ActionScript* de una forma muy sencilla.

El motivo por el que ha sido descartado el uso de *Silverlight* en el desarrollo del Proyecto de fin de carrera es que sería necesario realizar y mantener dos versiones de la aplicación. Una para los usuario de Windows y Mac (*Silverlight*) y otra para los usuario de Linux (*Moonlight*).

HTML5 ha sido descartada debido a que aún no es completamente operativa. De hecho la posibilidad de manejar la cámara *web* del usuario todavía no está disponible. Por lo tanto aunque en el futuro pueda ser una opción viable, de momento no es así.

A continuación se expone un gráfico comparativo de estas tres tecnologías.

	Flash	*Silverlight* / *Moonlight*	HTML5
Portabilidad	X		X
Acceso cámara *web*	X	X	
Estándares abiertos			X

2.3.- *Eyegrade*.

Es una herramienta desarrollada por el profesor D. Jesús Arias Fisteus. Está diseñada para corregir exámenes de tipo test mediante el uso de una cámara *web*. Dicha herramienta está desarrollada en *Python* y utiliza la biblioteca de código de visión artificial *OpenCV*.

Para llevar a cabo su función *Eyegrade*[1] necesita que se cumplan una serie de requisitos, los cuales son:

- ✓ Instalar el *software* de *Eyegrade* en el equipo del usuario.
- ✓ Un archivo con la configuración del examen, donde se indica el número de preguntas del examen, las soluciones, etc...
- ✓ Una cámara *web* compatible, con una resolución mínima de 640 x 480.
- ✓ La lista de alumnos convocados al examen, en caso de que el usuario quiera que *Eyegrade* identifique el número de identificación del alumno.
- ✓ Y lógicamente los exámenes para corregirlos.

Hay tres modos disponibles para el usuario, cuando éste ejecuta la herramienta. Uno de ellos, *search mode*, se encarga de realizar las correcciones mientras que otro, *review mode*, es el encargado de mostrar al usuario el resultado de la corrección. Éstos son los dos modos habituales en el funcionamiento de la herramienta.

Ademas hay un tercer modo adicional, *manual detection mode*, que solo se utiliza en muy raras ocasiones. Éste le permite al usuario identificar los límites del cuadro donde están las respuestas del alumno, para facilitar al programa la corrección de las mismas.

En el modo encargado de la corrección el usuario debe apuntar con la cámara *web* a la zona donde están las respuestas del alumno. Se debe incluir las casillas donde está escrito el identificador del alumno y las marcas situadas debajo, que son las que indican el tipo de examen a corregir.

La herramienta se encarga de ir realizando capturas de forma automática hasta que detecta correctamente toda esa información. En ese momento le pasa el control al modo encargado de mostrar el resultado de la corrección.

En este modo el usuario puede revisar y modificar la corrección hecha por la herramienta, tanto las respuestas como el identificador del alumno. La información se le muestra al usuario sobre impresionada encima de la imagen del examen que ha sido corregido. La información que se le muestra al usuario es:

- ✓ El número de identificación del estudiante.
- ✓ El número de exámenes previamente corregidos.
- ✓ Los símbolos de la corrección: un círculo verde para las respuestas correctas del alumno y uno rojo para las erróneas. En el caso de que la respuesta sea errónea también pinta un punto azul en la posición donde está la solución a esa pregunta.
- ✓ El número total de aciertos, fallos y respuestas en blanco del examen.
- ✓ El modelo del examen.

Por último cabe destacar las distintas salidas que ofrece la herramienta *Eyegrade* para que el usuario pueda ir almacenando los diferentes resultados de las correcciones. Las cuales son:

- ✓ Un archivo con los resultados, que contiene una línea para cada examen ya calificado. Cada línea contiene, entre otras cosas, el número de identificación del estudiante, el número de respuestas correctas e incorrectas, y la respuesta a cada pregunta del examen.
- ✓ Una imagen de cada examen ya calificado, en formato PNG. Las imágenes se pueden utilizar como prueba para mostrar a los estudiantes.

Capítulo 3.- Requisitos.

Como se ha explicado en el apartado 1.1, el objetivo de este Proyecto es implementar un cliente para la aplicación *Eyegrade* que se ejecute en navegadores *Web*. Concretamente, se plantean los siguientes requisitos:

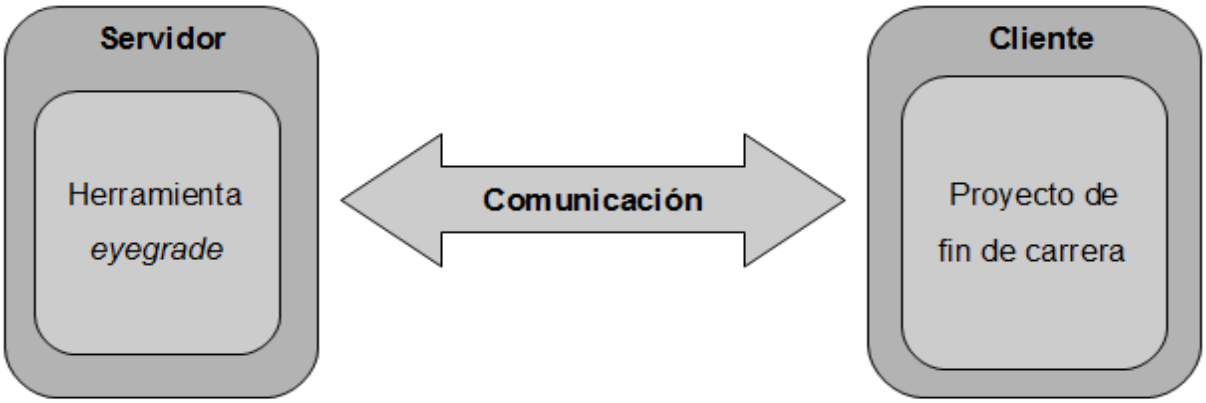
- ✓ El programa debe permitir al usuario elegir la cámara *web* que desee de entre las que tenga conectadas al equipo.
- ✓ Debe capturar imágenes de forma automática (a través de la cámara *web*) y enviarlas al servidor para su posterior procesado.
- ✓ Debe codificar las imágenes capturadas a monocromo (1 bit por *pixel*) para su envío al servidor.
- ✓ Debe posibilitar el uso de las herramientas de dibujo disponibles en Flash para pintar los símbolos de la corrección.
- ✓ Debe tener una función de video (no solo de captura), de esta manera se le permite al usuario buscar el enfoque correcto de una forma más cómoda.
- ✓ Tiene que ser capaz de cargar archivos locales (archivo de configuración, del cual se muestra un ejemplo en el apéndice 1, y lista de alumnos, hay un ejemplo en el apéndice 2) y enviarlos al servidor.
- ✓ Además, debe de permitir al usuario cargar una sesión, el apéndice 4 muestra un ejemplo de este archivo, ya existente para seguir trabajando sobre ella.
- ✓ También tiene que poder guardar datos, el tipo de archivo es el mismo que en el punto anterior, en el disco duro local, en la ubicación que el usuario indique.
- ✓ Por otra parte, ha de ser capaz de recibir respuestas, los tipos de respuestas se muestran en el apéndice 3, por parte del servidor y utilizar los datos encapsulados en las mismas.
- ✓ Debe permitir al usuario modificar interactivamente las respuestas detectadas por la herramienta *Eyegrade*, además de corregir el identificador de alumno

enviado por la herramienta.

- ✓ Debe funcionar correctamente en los navegadores *web* (Mozilla, Opera, Internet Explorer, etc.) y sistemas operativos (Linux, Windows, Mac OS) más habituales.

Capítulo 4.- Diseño de la aplicación.

4.1.- Arquitectura global del sistema.



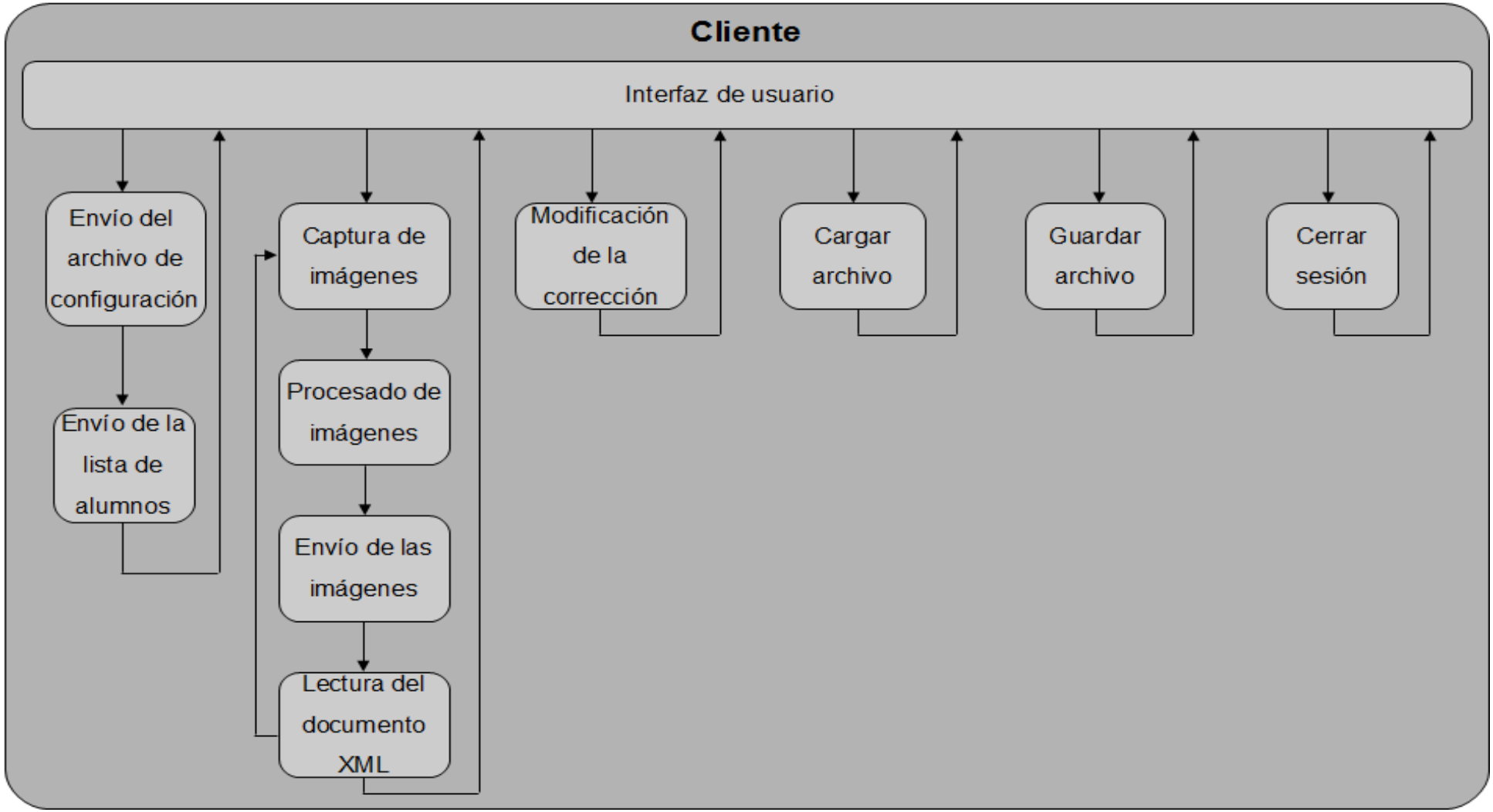
Éste es el esquema general del sistema. En él se aprecian las dos entidades que participan en el mismo: el programa desarrollado en este proyecto que actúa como cliente y la herramienta *Eyegrade* alojada en el servidor.

La comunicación entre ambos se realiza mediante el protocolo HTTP. El cliente encapsula en los mensajes de petición HTTP los datos que necesita *Eyegrade* para corregir los exámenes. El servidor envía el resultado del procesado al cliente en los mensajes de respuesta.

Por un lado, la función del cliente es la de conseguir la interacción con el usuario. Para ello se han creado una serie de pantallas en las que se van solicitando diferentes acciones al usuario para poder proceder con la corrección de los exámenes.

Por otro lado, el servidor se encarga de procesar las imágenes, corregir los exámenes y enviar de vuelta los datos de la corrección al cliente para que este los muestre por pantalla.

4.2.- Arquitectura de la aplicación cliente.



Dentro del apartado del cliente habría que destacar varios módulos independientes entre sí, que son los que vienen destacados en la figura anterior. A continuación se explica brevemente cada uno de ellos:

El módulo de envío del archivo de configuración necesita de la ejecución del módulo encargado de cargar los archivos. El usuario primero carga el archivo de configuración que tiene en su disco local y después a través de este módulo envía esos datos al servidor.

El módulo de envío de la lista de alumnos al igual que el módulo anterior, también necesita del módulo encargado de cargar los archivos para su correcto funcionamiento. Carga el archivo con la lista de alumnos y lo envía al servidor. Esta parte es opcional, el usuario no tiene por qué enviar esta información.

El módulo de captura de las imágenes se encarga de obtener las imágenes a través de la cámara *web* del usuario y mostrárselas por pantalla.

El módulo de procesado de imágenes trata las mismas antes de enviarlas al servidor. Ésto se hace para conseguir el mejor aprovechamiento posible del ancho de banda que tenga disponible el usuario en el lugar donde utilice esta aplicación.

Una vez se ha procesado la imagen, el módulo de envío de las mismas se encarga de mandárselas al servidor.

El módulo encargado de la lectura del documento XML procesa el mismo. Éste es enviado por el servidor como respuesta al envío de la imagen que hace el módulo anterior.

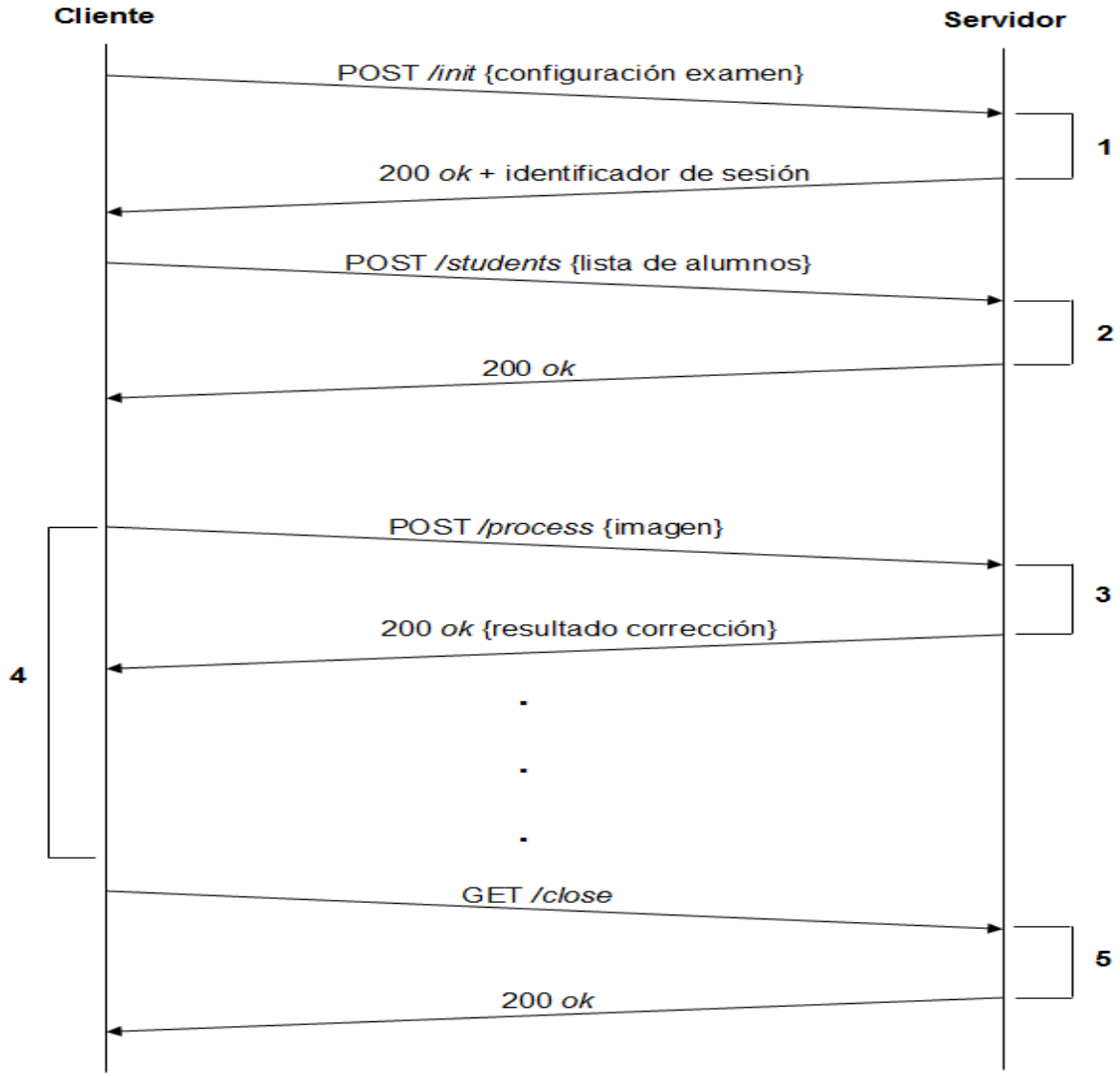
El módulo de modificación de la corrección le permite al usuario cambiar la respuesta enviada por el servidor y dejar el examen perfectamente corregido para su posterior almacenamiento.

El módulo de cerrado de la sesión envía un mensaje al servidor para indicarle que el usuario desea descartar la sesión actual.

El módulo encargado de cargar archivos locales abre un cuadro de dialogo que le permite al usuario poder cargar los mismos. Una vez cargados se pueden enviar al servidor o continuar trabajando con ellos (en el caso de cargar una sesión preexistente, ésto se explicará más en profundidad en capítulos posteriores).

Por último el módulo encargado de guardar archivos permite al usuario almacenar en su disco local el archivo con el resumen de la sesión.

4.3.- Protocolo de comunicación.



1.- Crear sesión, almacenar configuración de examen.
2.- Almacenar lista de alumnos.
3.- Procesado de imágenes
4.- Corrección de exámenes.
5.- Descartar sesión.

La comunicación con el servidor se hace mediante el protocolo HTTP. A continuación se detallan los tipos de peticiones que se utilizan durante la misma.

En la comunicación entre el servidor y la aplicación hay que seguir un orden preestablecido para que todo funcione correctamente. En la figura anterior se puede observar el orden en el que se realizan dichas peticiones.

Para poder iniciar una sesión entre ambos la aplicación debe enviar una petición HTTP POST al recurso "/*init*" del servidor. Ésta debe llevar un *Content-Type* = "*application/x-eyegrade-exam-config*". En el cuerpo de dicha petición se encapsulan los datos contenidos en el fichero de configuración, se puede ver un ejemplo de dicho archivo en el apéndice 1, para que el servidor sepa el tipo de examen que va a corregir. El servidor envía la respuesta indicado que ha recibido bien los datos y el número de identificador para toda la sesión.

Una vez establecida la comunicación entre ambas entidades el usuario puede enviar cualquiera de los otros tres tipos de mensajes incluidos en el gráfico anterior.

El usuario puede enviar o no la lista de alumnos. Este paso es imprescindible si quiere que el servidor reconozca el NIA y le devuelva el nombre del alumno al que pertenece el examen. La aplicación genera una nueva petición HTTP POST con un *Content-Type* = "*application/x-eyegrade-student-list*" y la envía al recurso "/*students*" del servidor. En el cuerpo de dicha petición se encapsulan los datos obtenidos del fichero con la lista de alumnos, se puede ver ejemplo del archivo con la lista en el apéndice 2. El servidor responde a esta petición con otro mensaje indicando que se ha recepcionado correctamente.

Cuando el usuario comience a enviar imágenes la aplicación genera una petición HTTP POST (por cada imagen) con un *Content-Type* = "*application/x-eyegrade-bitmap*" y la envía al recurso "/*process*" del servidor. Cada petición contiene en la parte de los datos la imagen codificada en binario. El servidor responde a esta petición enviando un documento XML en el cual indica si ha podido o no corregir la imagen, en el apéndice 3 se puede ver un ejemplo de cada tipo de respuesta.

En el caso de haber sido corregida el documento XML que envía el servidor como respuesta contiene:

- ✓ El campo donde indica que ha realizado la corrección.
- ✓ El número de fallos, aciertos, preguntas en blanco y dudosas.
- ✓ La respuesta del alumno a cada una de las preguntas y cuál es la opción correcta.
- ✓ Las coordenadas del centro de cada celda y la longitud de la diagonal de cada

una de ellas.

De esta manera se facilita el pintar el símbolo que corresponde en cada una de ellas (un círculo donde se haya detectado la respuesta del alumno y un punto en la casilla donde está la respuesta correcta en caso de que no coincida con la del alumno).

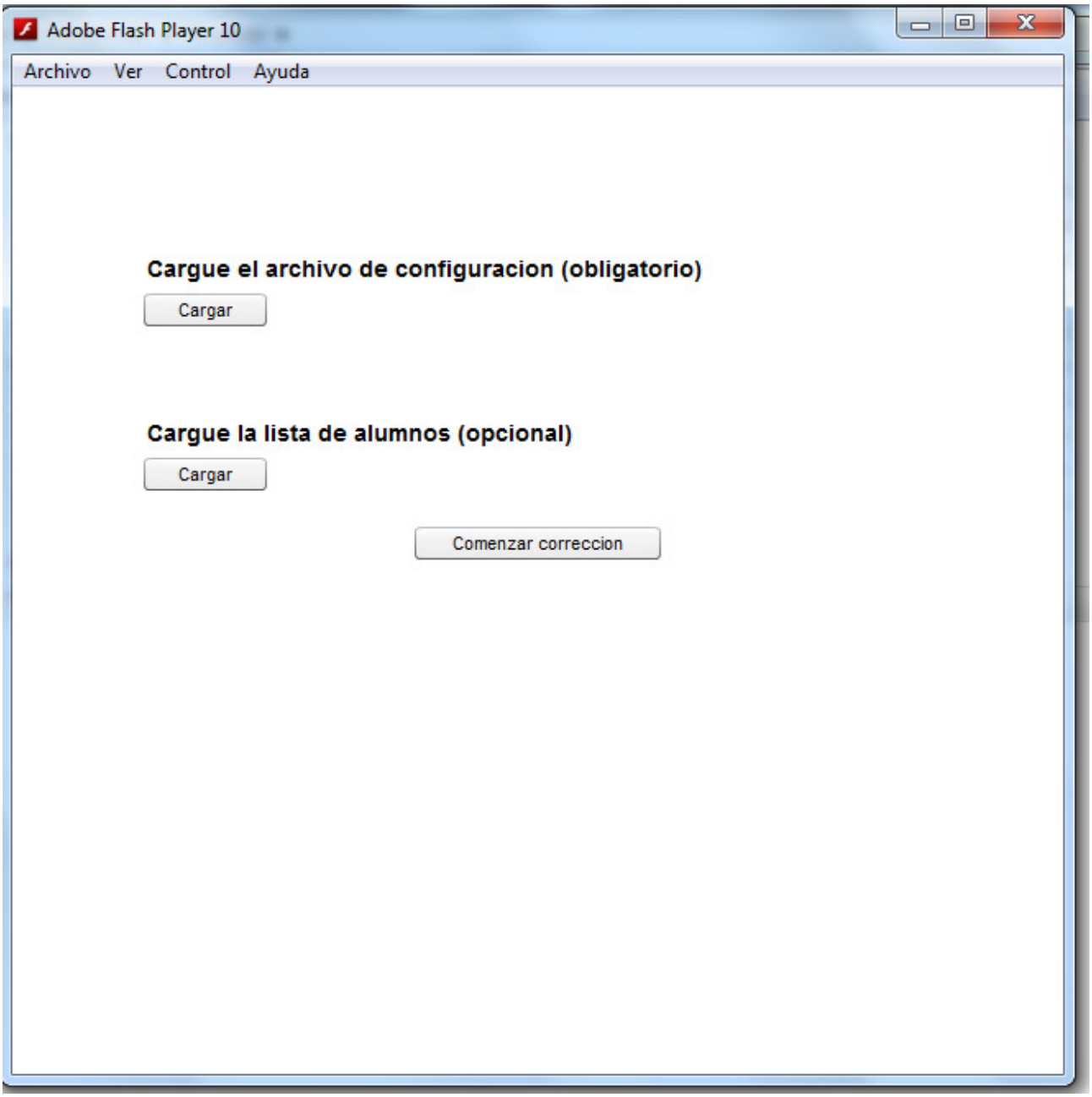
Por último, cuando el usuario decida poner fin a una sesión deberá enviar al servidor una petición GET al recurso "/*close*". Ésto le indica al servidor que se desea cerrar la sesión. El servidor envía también una confirmación de que ha recibido correctamente la petición y ha cerrado la sesión. Esto le permite al usuario poder empezar con una nueva sesión si así lo desea.

4.4.- Vistas de la aplicación e interacción con el usuario.

La aplicación consta de 4 vistas, con las que va a interactuar el usuario, las cuales son:

- ✓ Pantalla inicial. En ella se van a cargar los archivos de configuración.
- ✓ Pantalla sesión. En esta vista se elige si cargar una sesión existente o empezar con una nueva.
- ✓ Pantalla principal. En ella es donde se desarrolla el grueso del programa, es donde se capturan los exámenes y se envían para su corrección.
- ✓ Pantalla salir. En esta vista se da la posibilidad al usuario de guardar los resultados de las correcciones antes de cerrar la aplicación.

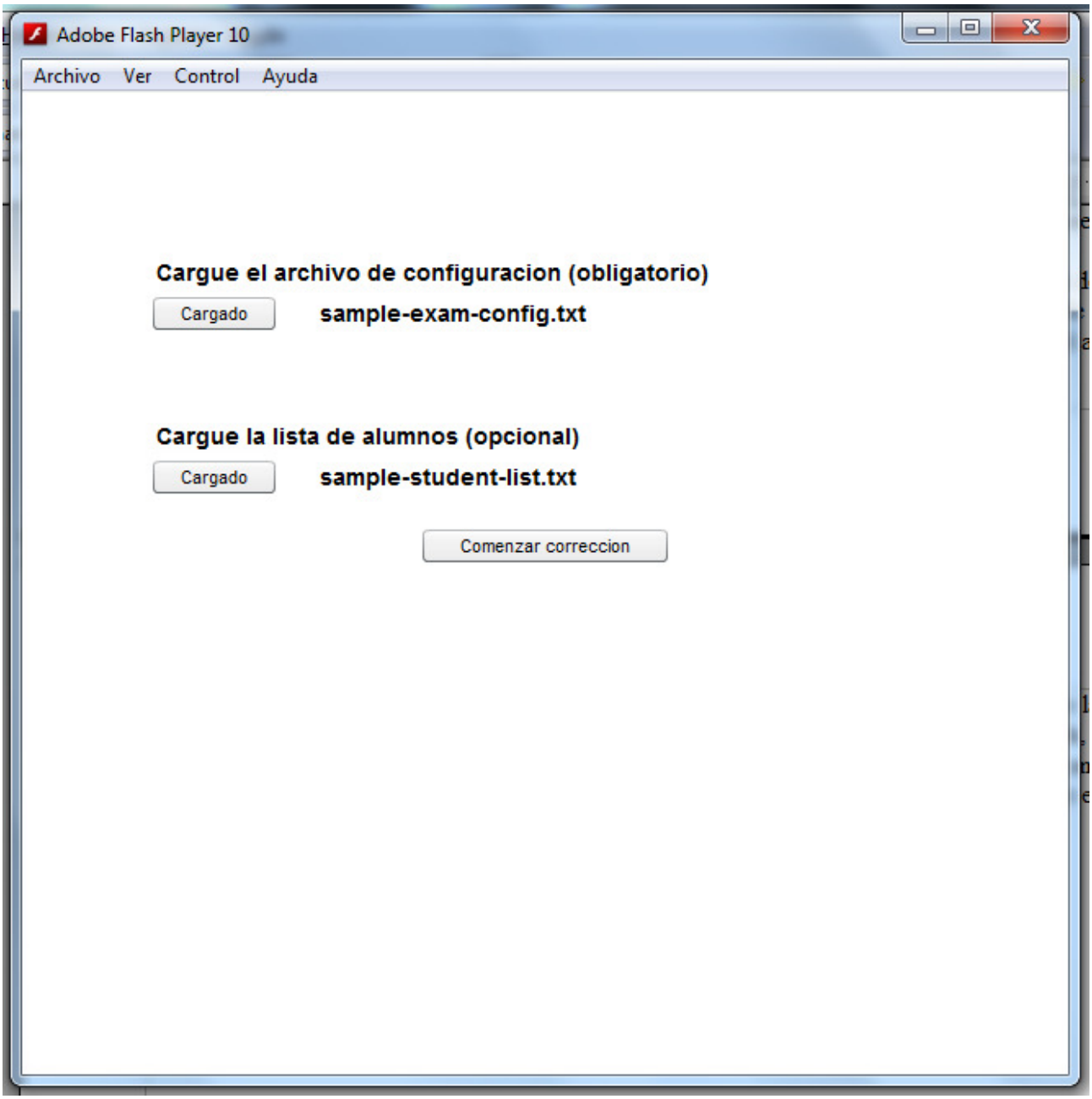
4.4.1.- Pantalla inicial.



En esta pantalla el usuario puede ver dos etiquetas que le dan instrucciones sobre cuáles son las acciones a llevar a cabo. En este caso debe cargar un archivo de configuración y si lo desea también puede cargar la lista de alumnos.

Una vez cargado el archivo de configuración el usuario ya podría empezar la corrección del examen independientemente de si ha cargado la lista o no. En caso contrario al dar al botón de comenzar corrección saldría un mensaje emergente (un *popup*) que le indica que debe cargar el archivo de configuración.

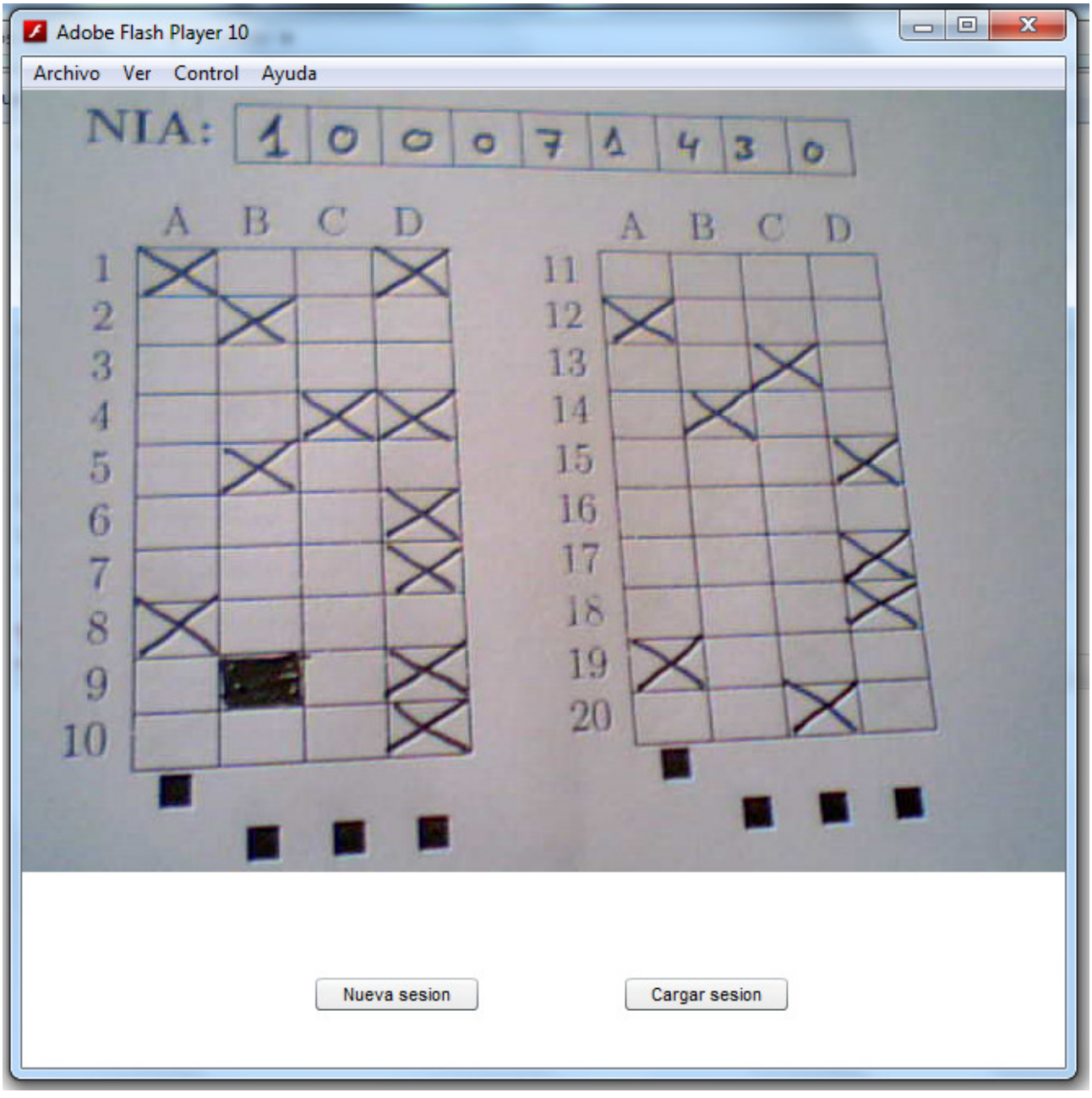
Cuando el archivo está cargado y listo para ser enviado al servidor la apariencia de la pantalla inicial varía levemente, para indicar al usuario que el archivo esta correctamente cargado. Cambia la etiqueta del botón de “Cargar” a “Cargado” (tanto para el archivo de configuración como para la lista de alumnos) y al lado del botón aparece el nombre del archivo seleccionado por el usuario, quedando la apariencia de la siguiente manera:



Una vez estén todos los datos necesarios cargados, el usuario ya puede dar al botón de comenzar la corrección lo cual le lleva a la siguiente pantalla. En ella va a poder elegir entre comenzar una nueva sesión o continuar desde una ya existente.

La idea es que cada examen se guarde en una única sesión. De este modo quedan almacenados todos los datos de las correcciones de los alumnos en un único fichero en el disco local del usuario. Ésta parte será explicada en profundidad más adelante, en la pantalla de cerrar la sesión.

4.4.2.- Pantalla de sesión.



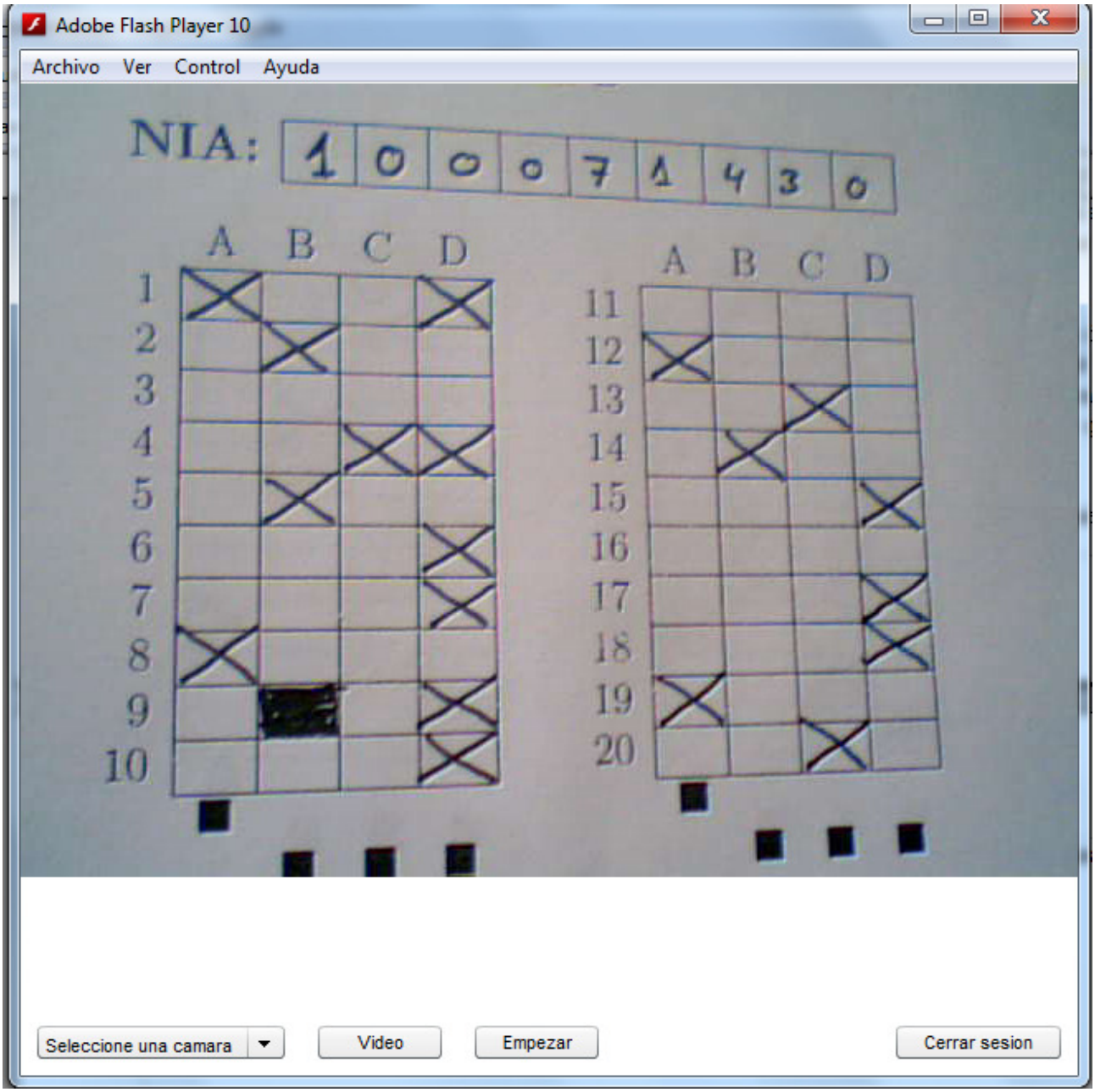
En esta pantalla se observa por primera vez lo que está captando la cámara *web,* además de dos botones cuya finalidad es establecer el tipo de sesión con la que queremos trabajar.

En el caso de ser una nueva sesión el programa genera una cadena de texto vacía. Ésta es asignada a la variable donde vamos a ir almacenando los resúmenes de cada una de las correcciones para ese examen.

En caso de que el usuario quiera seguir trabajando con una sesión ya existente en su disco local, tiene que pulsar “Cargar sesión”. Este botón abre un cuadro de dialogo que permite al usuario seleccionar el archivo a cargar. Una vez seleccionado, el programa carga los datos en la variable de texto que va a ir almacenando los resultados del resto de correcciones de esa sesión.

Cuando el usuario elija el tipo de sesión que quiere utilizar la aplicación pasa a la siguiente pantalla.

4.4.3.- Pantalla principal.



En esta pantalla se pueden observar tres botones y un *combo box.* Ésta es la pantalla más importante del programa y con la que más tiempo va a trabajar el usuario.

Lo primero cabe destacar que cuando se arranca la aplicación el programa obtiene una lista de las cámaras conectadas al sistema y carga sus nombres en el *combo*. En él el usuario puede seleccionar la que le resulte más conveniente para realizar las capturas. Por defecto siempre sale en pantalla una de las cámaras disponibles (siempre se carga la primera que se encuentra). Ajustamos la calidad del vídeo para que la resolución sea la máxima que nos permita la cámara *web*. Dicho ajuste se cambia siempre que el usuario

elija cambiar de cámara a través del *combo box* ya que no tienen porqué tener la misma resolución todas las que estén conectadas al sistema.

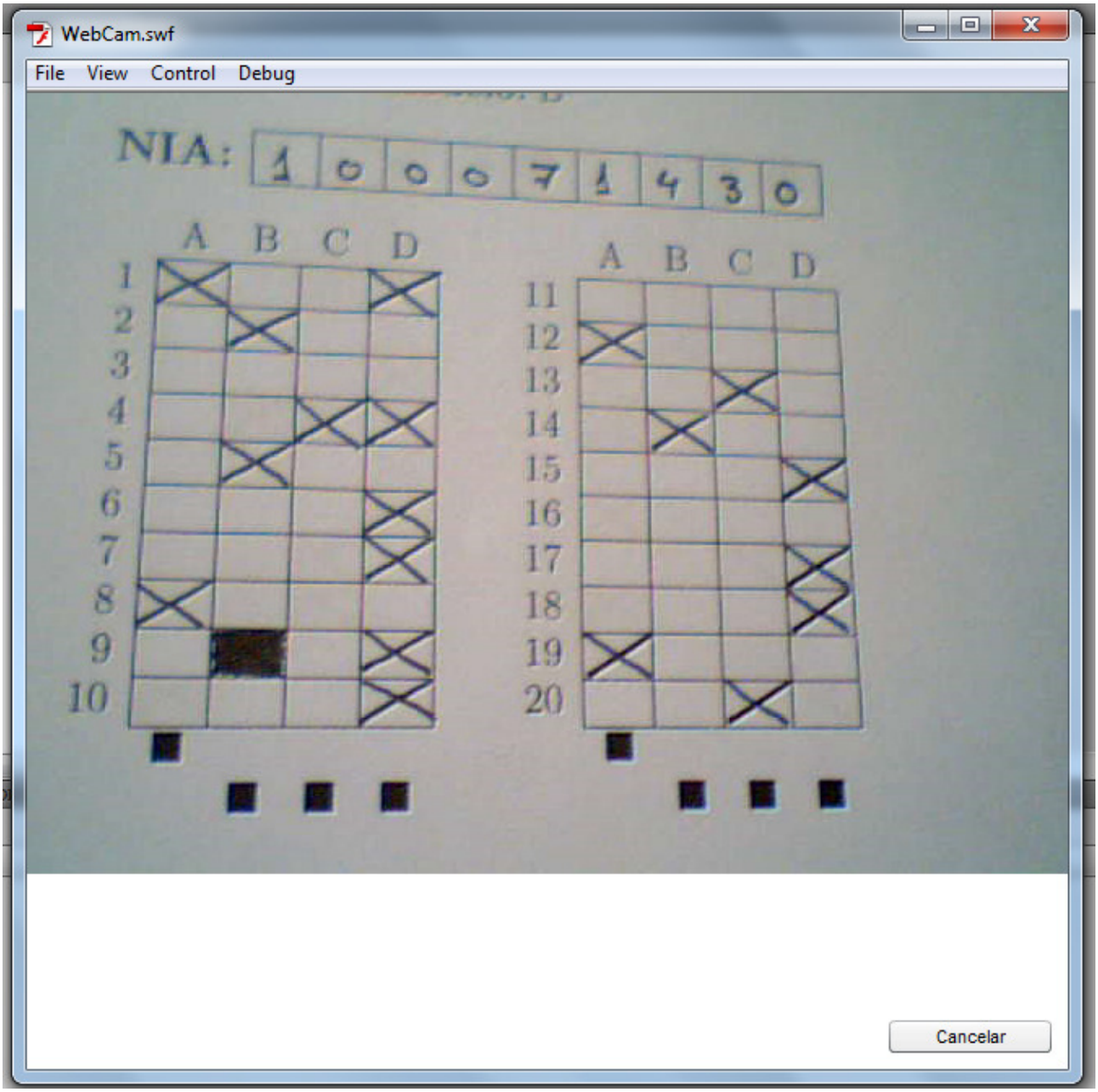
El siguiente botón que nos encontramos es el que pone “Video”. Este botón permite al usuario enfocar correctamente la imagen antes de empezar con una nueva captura. Para ello se le muestra al usuario la imagen captada por la cámara *web* de forma continua, para que pueda colocarla en la posición óptima antes de empezar a enviar imágenes al servidor.

Una vez colocada la cámara *web* en su posición, el usuario tiene que pinchar en el botón “Empezar”. Este botón es el que activa el grueso del programa, lo primero que sucede cuando el usuario le pulsa es que la etiqueta del botón pasa de “Empezar” a “Siguiente”. Después desaparecen los botones y el *combo* que se muestran en la pantalla anterior y aparece un botón de “Cancelar”. El programa empieza a capturar las imágenes para enviarlas al servidor, si el usuario quiere cancelar este proceso solo debe pinchar en el botón.

El objeto de visualización pasa ahora a mostrar las imágenes capturadas. Éstas serán enviadas al servidor mediante peticiones HTTP POST con las imágenes codificadas en binario. La respuesta a estas peticiones es un documento XML que envía el servidor indicando si el examen ha podido ser corregido o no. En caso de que haya sido corregido, en dicho documento XML vendrá toda la información necesaria para poder pintar la corrección sobre la imagen que está viendo el usuario.

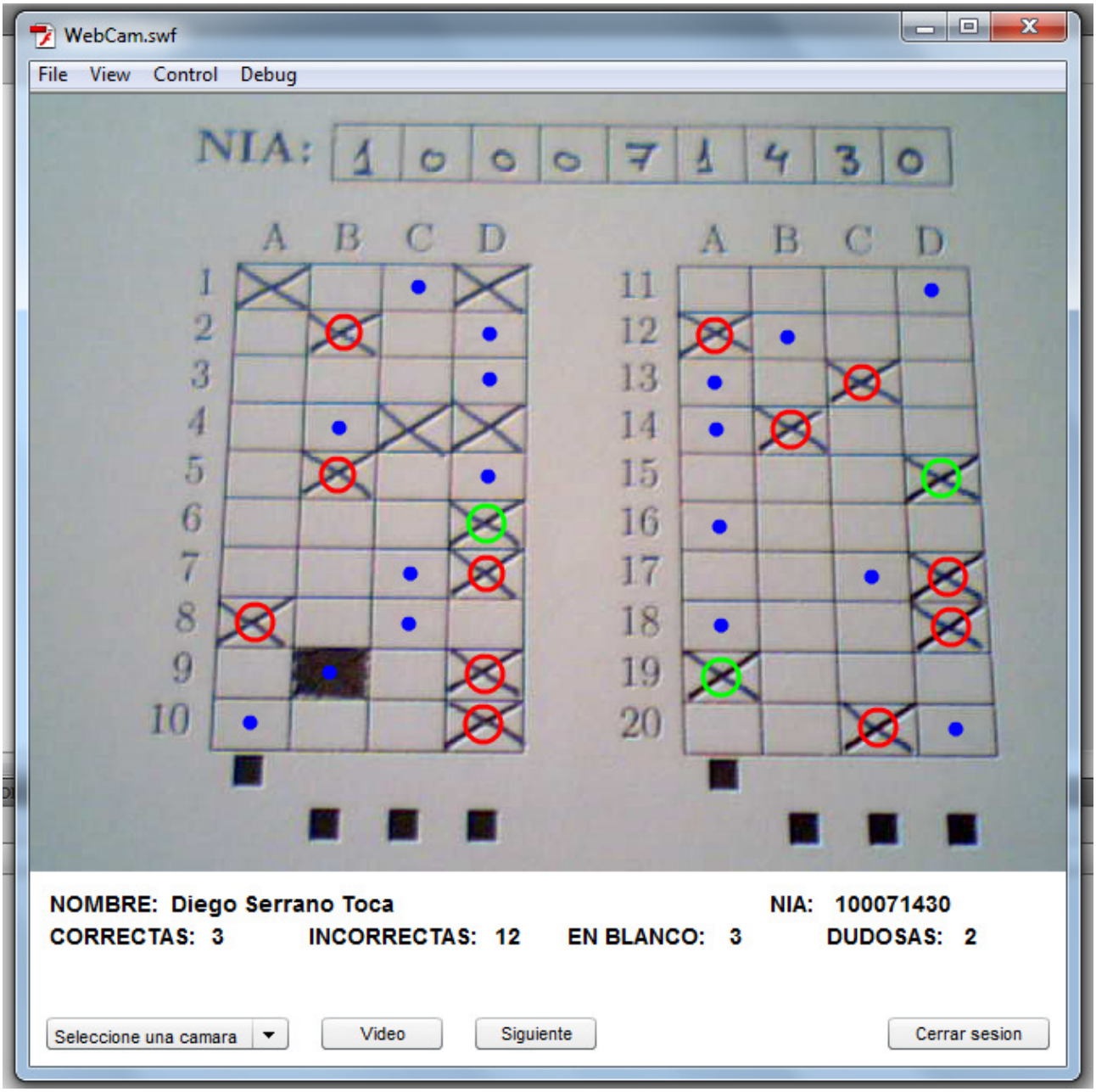
Previamente a enviar dicha imagen al servidor está la parte que se encarga de procesarla hasta reducirla a datos binarios. Esto es así porque la herramienta *Eyegrade* del servidor procesa imágenes en blanco y negro. Por lo tanto es más adecuado hacer la conversión en el cliente antes del envío, ya que conseguimos un ahorro significativo en el ancho de banda consumido.

Todo este proceso es completamente transparente para el usuario ya que éste solo ve la pantalla donde se van mostrando las imágenes que se están enviando al servidor.



Como se ha indicado antes este proceso se repite de forma automática hasta que el servidor envíe una corrección del examen en formato XML como respuesta a la petición HTTP POST que le envía el programa.

Cuando el programa recibe el documento XML el siguiente paso es procesar los datos para poder pintar sobre la imagen la corrección hecha por el servidor. Para ello se han creado diferentes métodos. Uno se encarga de leer la información directamente del documento XML para obtener las coordenadas donde se deben ir pintando los símbolos, así como el nombre del alumno y el número del NIA. Toda esta información se le muestra al usuario en la siguiente pantalla.



Cuando el documento XML indica que en una celda ha encontrado una respuesta del alumno, el programa llama al método que se encarga de pintar un círculo sobre la imagen. Dicho método necesita las coordenadas del centro de la celda, así como la diagonal de la misma para adecuar la posición y el radio del círculo. Si la respuesta es correcta el programa pinta un círculo verde, en caso de que sea incorrecta pinta uno rojo.

En el caso de que dicha respuesta no coincida con la correcta a esa pregunta, el programa llama al método que se encarga de pintar un punto azul sobre la imagen. Éste hace dicha marca sobre la casilla que se corresponde con la posición correcta para esa pregunta.

Una vez está la imagen corregida a disposición del usuario, éste vuelve a ver los botones que le habían desaparecido al iniciar las capturas. Por lo tanto ahora el usuario vuelve a disponer de todas las opciones que tenía cuando empezó la sesión más otra adicional, que es la de poder modificar la corrección que le ha enviado el servidor. Es decir podría volver a seleccionar otra cámara, poner el programa a funcionar en modo video, hacer una nueva captura o cerrar la sesión.

Para modificar la corrección enviada por el servidor, el usuario simplemente debe pinchar sobre la imagen. El programa comprueba si hay o no un símbolo en esa casilla. Si lo hay, mira el tipo de símbolo que esté pintado sobre ella. El programa realiza una acción distinta para cada caso, estas acciones son:

- ✓ Si no hay ningún símbolo en esa casilla:

El programa pinta un círculo sobre la casilla donde ha pulsado el usuario y en caso de que hubiera otra marca de respuesta del alumno para esa pregunta, el programa la borraría, respetando el código de colores previamente especificado.

- ✓ Si hay un punto en esa casilla:

El programa borra el punto azul de esa casilla y lo sustituye por un círculo verde. Es decir cambia el símbolo de respuesta correcta por el símbolo de respuesta del alumno. Además en caso de que hubiera otra marca de respuesta del alumno para esa pregunta, el programa la borraría.

- ✓ Si hay un círculo en esa casilla:

El programa borra esa marca de respuesta del alumno dejando la pregunta como sin contestar por el alumno. Si además coincide que la casilla donde estaba el círculo es la respuesta correcta, el programa pinta un punto azul sobre esa casilla. En caso contrario no pintaría nada.

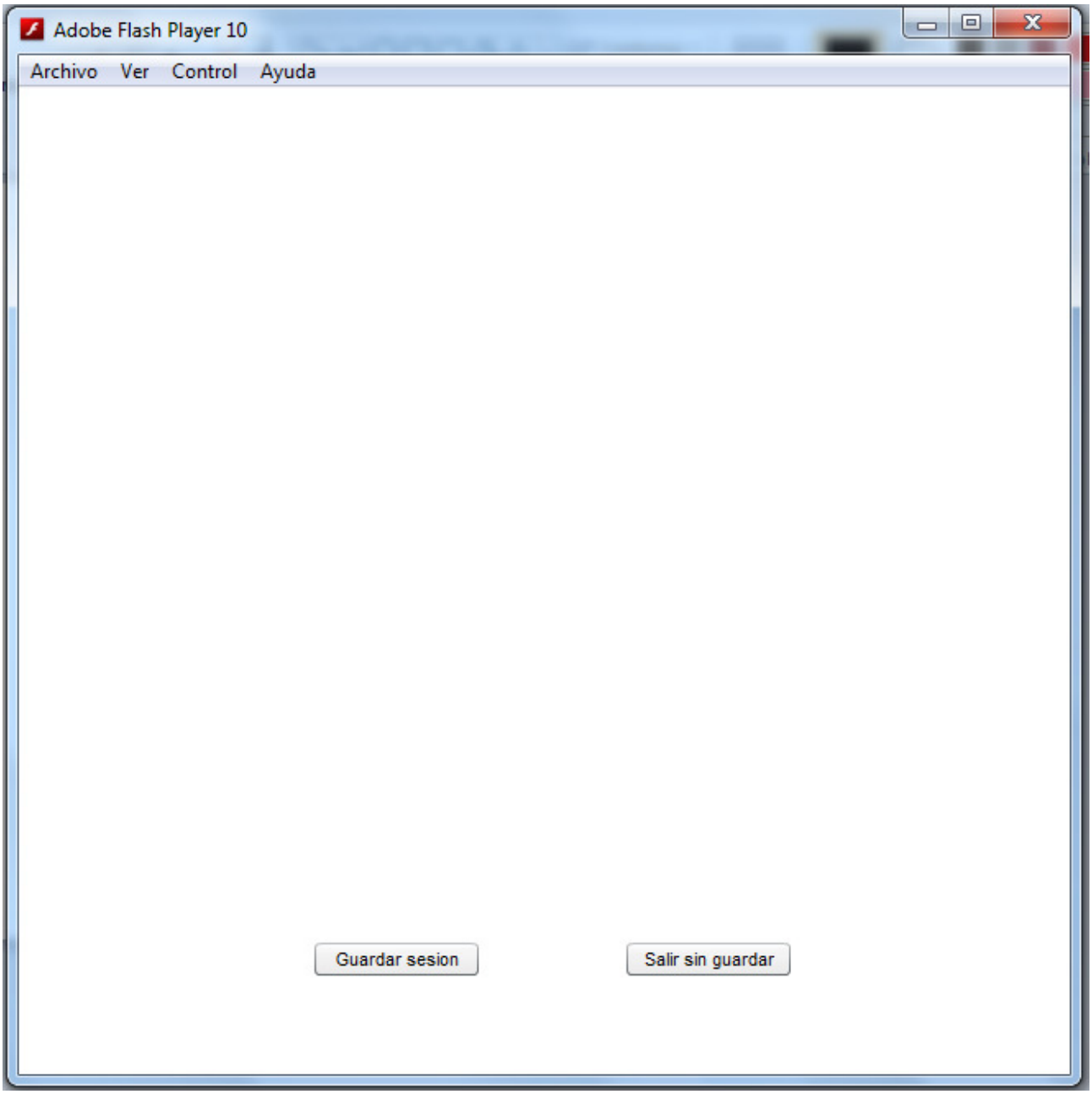
Por otro lado, además de permitir al usuario modificar la corrección automática del servidor, también se le permite modificar los datos correspondientes al NIA y al nombre del alumno al cual corresponde el examen. Para ello el usuario lo único que tiene que hacer es pinchar sobre la etiqueta donde se van a pintar dichos datos y de esta forma podrá sobrescribirlos y modificar el contenido de los mismos. Esto hay que hacerlo tanto

en el caso de que el servidor haya enviado la información como si no ha reconocido el NIA.

Una vez el usuario ha dado por buena la corrección dispone de las opciones previamente detalladas. Siempre que el usuario pinche en “Cerrar sesión”, “Siguiente” o “Video” los datos de la corrección anterior (en caso de que exista) se almacenarán de forma temporal en una variable. Esto se hace así por si el usuario decide que quiere guardar las correcciones de toda la sesión en un archivo en su disco local. De esta forma cada corrección se almacena una única vez en el archivo.

El programa continúa su funcionamiento normal hasta que el usuario decida que quiere cerrar la sesión, bien porque ha terminado de corregir todos los exámenes de esa sesión o bien porque no quiera seguir con la corrección. En ese momento la pantalla que ve el usuario cambia y tiene la siguiente apariencia.

4.4.4.- Pantalla salir.



Esta nueva pantalla contiene dos botones. Sirven para que el usuario pueda salir de la sesión que había iniciado previamente. Tiene la opción de guardar los datos de la sesión o simplemente cerrarla sin guardar los datos.

En caso de que el usuario quiera guardar los datos de la sesión el programa comprueba que efectivamente haya datos que guardar. Si el usuario ha empezado una nueva sesión pero no ha corregido ningún examen no tiene sentido que guarde ningún dato. En caso de que haya abierto una sesión existente y tampoco haya corregido ningún examen adicional sí que podría guardar los datos aunque éstos serian exactamente los mismos que tenía antes de empezar.

En cualquiera de los dos casos una vez el usuario haya cerrado la sesión, el programa envía una petición HTTP GET al servidor indicando el cierre de la sesión y vuelve a mostrarle la pantalla inicial para que pueda volver a empezar con otra corrección.

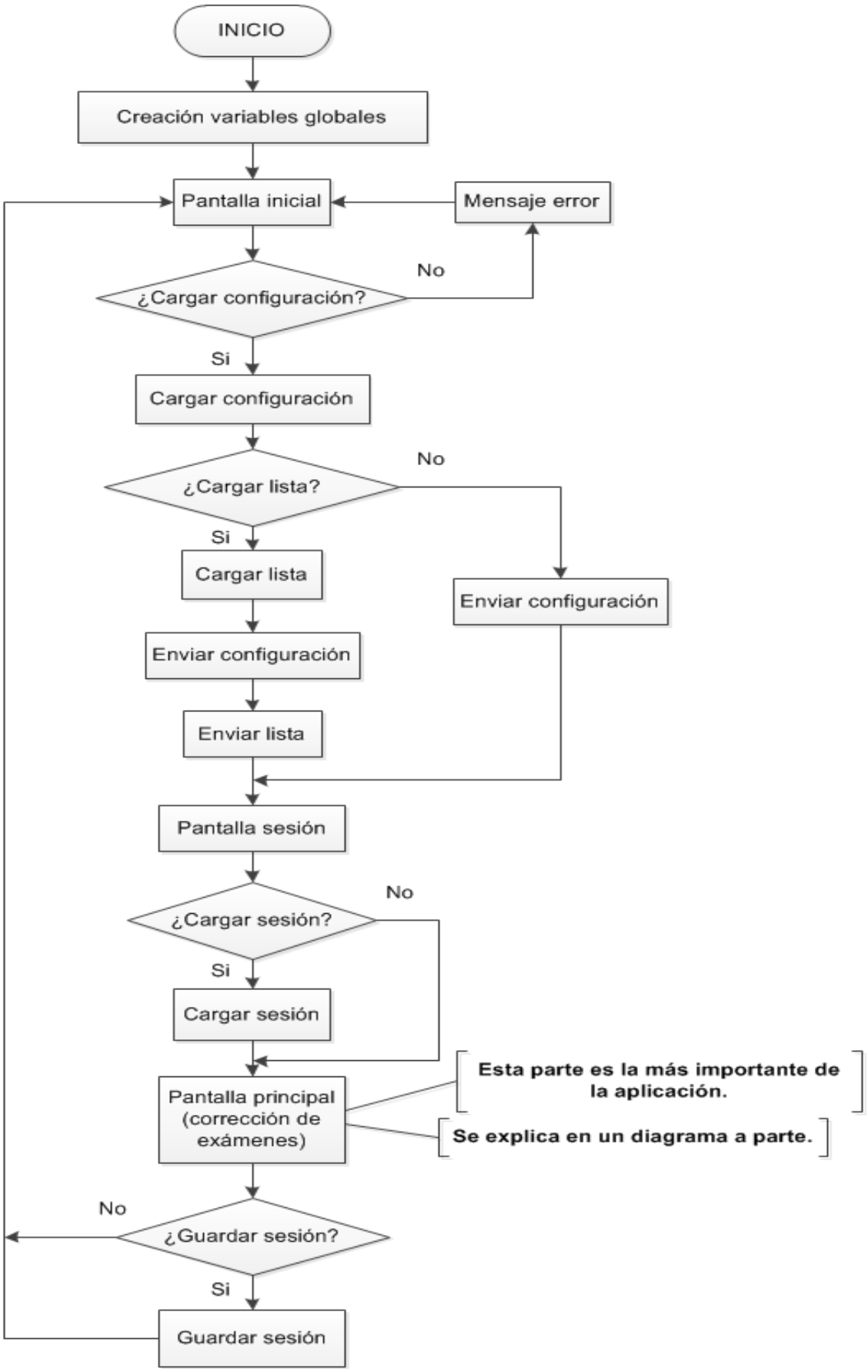
4.5.- Flujo de la aplicación.

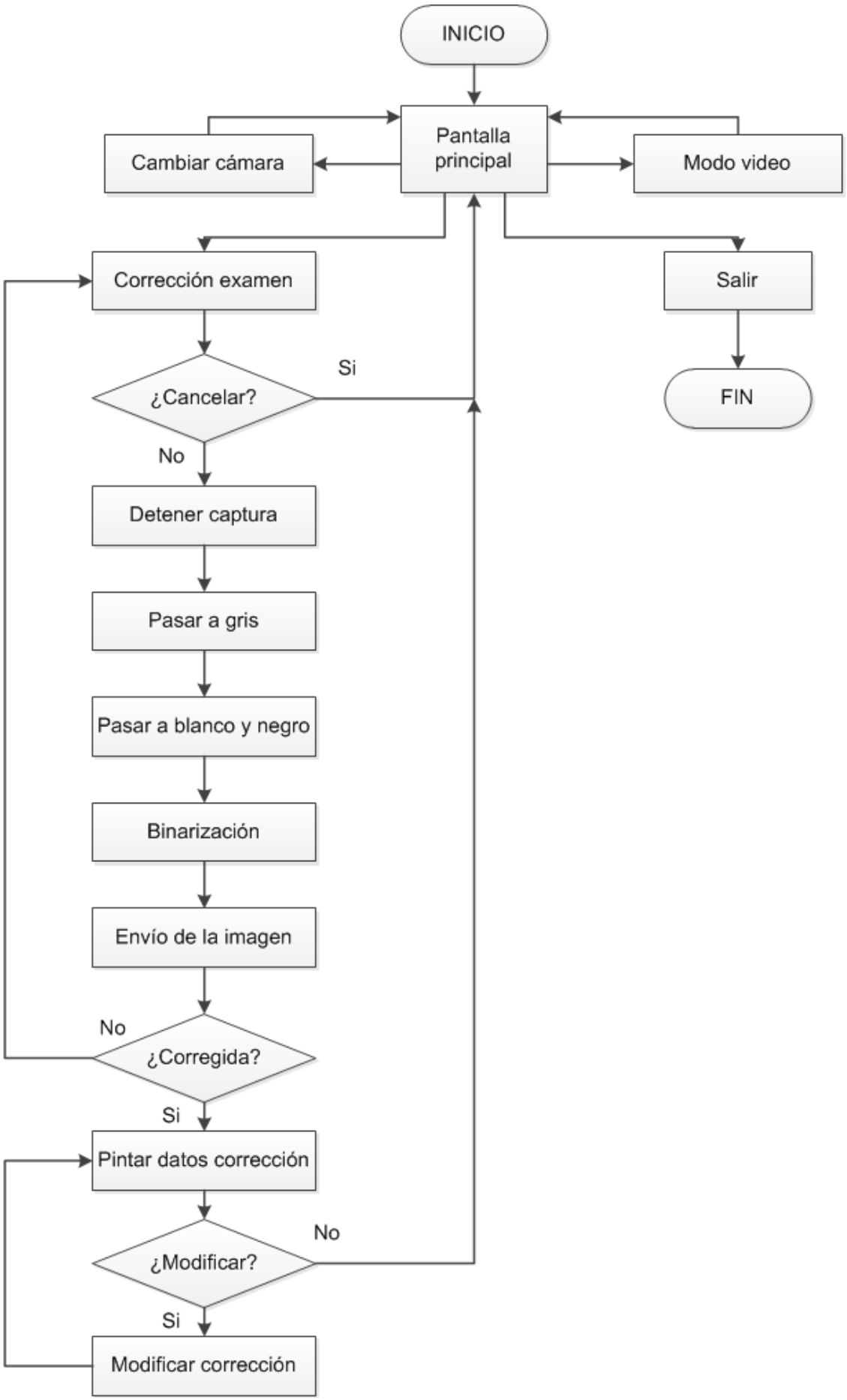
A continuación se expone un diagrama de flujo donde se detallan las diferentes acciones llevadas a cabo por la aplicación. Se muestra también la interacción entre las diferentes pantallas que la componen.

Dado que el módulo encargado de la corrección de exámenes es el más importante dentro de la aplicación su funcionamiento se explica en un diagrama aparte.

En el primer diagrama, se observa el orden en el que se ejecuta la aplicación desde que se abre por primera vez hasta que se cierra.

Mientras que en el segundo, se detalla todo lo que ocurre mientras el usuario está en la pantalla principal. Explicando que procesos activa cada uno de los botones.


INICIO
Creación variables globales
Pantalla inicial
Mensaje error
No
¿Cargar configuración?
Si
Cargar configuración
No
¿Cargar lista?
Si
Cargar lista
Enviar configuración
Enviar configuración
Enviar lista
Pantalla sesión
No
¿Cargar sesión?
Si
Cargar sesión
Pantalla principal (corrección de exámenes)
Esta parte es la más importante de la aplicación.
Se explica en un diagrama a parte.
No
¿Guardar sesión?
Si
Guardar sesión


INICIO
Pantalla principal
Cambiar cámara
Modo video
Corrección examen
Salir
¿Cancelar?
Si
FIN
No
Detener captura
Pasar a gris
Pasar a blanco y negro
Binarización
Envío de la imagen
No
¿Corregida?
Si
Pintar datos corrección
No
¿Modificar?
Si
Modificar corrección

Capitulo 5.- Implementación.

En este capítulo se van a ir destacando los aspectos más relevantes del código desarrollado en el programa. Las funcionalidades que se van a explicar a continuación son:

- ✓ Envío de ficheros locales al servidor.
- ✓ Listado de las cámaras y selección de una de ellas.
- ✓ Captura de la cámara *web* (modo imagen y modo video).
- ✓ Codificación de las imágenes en monocromo.
- ✓ Binarización de la imagen.
- ✓ Envío de las imágenes al servidor.
- ✓ Análisis del XML devuelto por el servidor.
- ✓ Pintado de una capa sobre la imagen capturada.
- ✓ Detección de las celdas de la imagen donde el usuario pincha.
- ✓ Cargado de ficheros locales en la aplicación.
- ✓ Guardado de ficheros en la maquina local.
- ✓ Cerrado de la sesión entre el cliente y el servidor.
- ✓ Creación de mensajes *PopUp*.

5.1.- Envío de ficheros locales al servidor.

El primer paso del mismo consiste en cargar los archivos tanto de configuración como la lista de los alumnos (aunque este segundo es opcional para el usuario). En ambos casos el funcionamiento del programa sigue el mismo esquema.

Para realizar esta funcionalidad se han utilizado varias funciones. La explicación será sobre las usadas para el archivo de configuración ya que para la lista son similares. En caso de que haya alguna diferencia notable se comentará en qué consiste.

Antes de empezar con las funciones es necesario crear algunas variables para poder utilizarlas en varias de ellas. Dichas variables son:

archivoURLL:*URLLoader*.

archivoURLR:*URLRequest*.

archivo:*FileReference*.

Función cargarArchivo.

Este es la función que se activa cuando el usuario pulsa el botón de cargar el archivo de configuración. Crea el objeto *FileReference* (que ya había sido definido previamente).

A dicho objeto se le añaden tres escuchadores: Uno para cuando el usuario pulse seleccionar en el cuadro de diálogo que se le va a abrir para elegir el archivo de configuración, otro para cuando pulse cancelar en el mismo cuadro de diálogo y otro para cuando en el caso de haber seleccionado un archivo se complete la carga de los datos del mismo.

Por último se invoca al método *browse*() de *FileReference* para que se abra el cuadro de diálogo, donde el usuario pueda seleccionar el archivo entre los que estén disponibles en su equipo local.

Función archivoSeleccionado.

Esta función se invoca cuando el usuario pulsa cancelar o seleccionar en el cuadro de diálogo donde se elige el archivo de configuración.

Lo primero que hace esta función es comprobar si se ha pulsado el botón seleccionar o el de cancelar, ya que la acción a realizar en cada caso es diferente.

En caso de que se haya pulsado el botón de seleccionar, el programa pone en una etiqueta de texto el nombre del archivo seleccionado, para indicarle al usuario el archivo que se va a enviar al servidor, e invoca al método *load*() de *FileReference* para iniciar la carga de los datos.

En caso contrario, es decir, que se haya pulsado la tecla de cancelar, el programa limpia la etiqueta de texto donde iría el nombre del archivo para indicarle al usuario que no se ha cargado ningún archivo aún.

Función archivoCargado.

Se invoca, mediante un evento, a esta función cuando se completa la instrucción *load*() previamente relatada. Ésto implica que cuando se llega a este punto el programa ya dispone de los datos del archivo en el objeto *FileReference* y por lo tanto ya se puede crear la petición HTTP que se va a enviar al servidor. Para ello se siguen los siguientes pasos:

- ✓ Se crea la cabecera de la petición, que es donde se le indica el Content-Type a utilizar en este caso *application/x-eyegrade-exam-config* (en el caso de la lista de alumnos seria *application/x-eyegrade-student-list*).

- ✓ Después se crea una variable de texto donde se va a guardar la cadena con la dirección a la que hay que enviar la petición. En el caso del archivo de configuración sería http://plato.it.uc3m.es:3999/init, mientras que en el caso de la lista de alumnos sería http://plato.it.uc3m.es:3999/students.

- ✓ Se crea un nuevo objeto *URLRequest* con dicha dirección. Este objeto había sido definido previamente para poder utilizarlo en más de una función. El programa pasa los datos que tiene disponibles en el *FileReference* al *URLRequest*, indicándole que el método de la petición va a ser el POST y le añade la cabecera.

Los datos se enviarán cuando el usuario quiera empezar la corrección, no cada vez que cargue un archivo. Para su envío se utilizan las siguientes funciones:

Función enviarConfiguracion.

Esta función hace uso de la variable archivoURLL definida anteriormente. Por lo tanto el siguiente paso es crearla con los valores necesarios para poder enviar la información del archivo de configuración. Al crear este tipo de variable los datos son enviados al servidor mediante una petición HTTP con los parámetros asignados al definir la variable archivoURLR.

Además de enviar los datos en esta función también se le añaden dos escuchadores a la variable archivoURLL. Uno para que lance un evento cuando finalice la petición, esto es cuando el servidor recibe la petición y devuelve un mensaje con el identificador de sesión almacenada en una *cookie* que se va a utilizar durante toda la comunicación entre el cliente y el servidor. Esta parte es muy significativa, ya que no se puede enviar ningún tipo de mensaje antes de recibir la respuesta del primero. Por lo tanto de este modo el programa se asegura de que no se envíe nada más al servidor hasta disponer del identificador de sesión.

El otro escuchador que se añade a esta variable es uno del tipo *ioError* para controlar que la comunicación entre ambas entidades funcione correctamente.

Cuando se lance el evento de envío completado la siguiente función en entrar en juego es:

Función respuestaCompleta.

Esta función tiene una única utilidad, la de comprobar si el usuario ha cargado algún archivo con la lista de alumnos o no. En caso de que no haya cargado ningún archivo no hace nada, pero si el usuario ha cargado la lista llama a la función que se encarga de enviar la lista al servidor que es enviarAlumnos.

La función de enviarAlumnos es prácticamente igual que la de enviarConfiguracion, salvo porque en este caso no se añade el escuchador de envío completo ya que no es necesario.

El escuchador que se añade para comprobar el buen funcionamiento de la comunicación simplemente llama a otra función. Ésta se encarga de sacar un mensaje por

pantalla. Para lo cual necesita recibir el texto correspondiente, en este caso sería "Error al enviar datos al servidor".

5.2.- Listado de las cámaras y selección de una de ellas.

Se ha habilitado un *combo box* para que el usuario seleccione una cámara entre las disponibles, en él se van a cargar todas las cámaras que tenga el usuario conectadas a su equipo en el momento en que accede al programa. Ésto se consigue haciendo uso de la clase *Camera* de *ActionScript* que contiene todos los nombres de las cámaras conectadas al equipo [14].

Cada vez que el usuario cambia de cámara hay una función que se encarga de cargar la que ha seleccionado y le muestra las imágenes captadas por la misma. Asigna las propiedades de la cámara para darle los parámetros correctos, esto es, una resolución de 640 x 480 *pixeles*. Se indica al objeto que controla la cámara que obtenga la máxima calidad posible de ella.

5.3.- Captura de la cámara *web* (modo imagen y modo video).

Modo imagen.

Para capturar una imagen de la cámara *web* se crea un objeto *BitmapData*. Mediante el método *draw*() de dicho objeto se carga la imagen desde el objeto Video, a través del cual se ve qué es lo que está capturando la cámara.

Para hacer que el usuario vea la imagen que se va a enviar al servidor, hay que crear un objeto *Bitmap* y asociarlo al objeto de visualización. En realidad, la imagen que ve el usuario no es exactamente la que llega al servidor. Antes de su envío se procesa la imagen, pero sí que contiene la misma información útil.

Modo video.

Para hacer que el usuario vea la captura de la cámara *web* se crea un objeto *Sprite*. Se asocia el objeto Video creado previamente con éste y posteriormente el *Sprite* con el objeto de visualización para que el usuario disponga de la imagen en modo continuo.

5.4.- Codificación de las imágenes en monocromo.

Este apartado contiene varias funciones que se han desarrollado para simplificar el problema y hacer que sea lo más modular posible.

El primer paso es pasar la imagen disponible en el objeto *BitmapData,* a escala de gris. Para ello se crea un nuevo objeto *BitmapData* con el método *clone*() del primero. Este paso es imprescindible para hacer que este proceso sea transparente para el usuario, ya que si se hiciera el tratamiento sobre el *BitmapData* original los cambios serian visibles para el usuario. Por lo tanto hay un objeto *BitmapData* asociado a un *Bitmap* que es el que se va a mostrar al usuario. El otro objeto *BitmapData* que es el que va a ir sufriendo todo el proceso de transformación para poder ser enviado al servidor.

A partir de aquí siempre que se hable del objeto *BitmapData* se hace referencia al que va a ser tratado, a no ser que se indique lo contrario.

Una vez está el objeto *BitmapData* listo para su manipulación, el primer paso es crear un objeto del tipo *ColorTransform*. En él se van a indicar los factores por los que se va a multiplicar cada una de las componentes de color de cada *pixel*. En este caso se utiliza una ponderación del siguiente tipo: *redMultiplier* = 0.30, *greenMultiplier* = 0.59 y *blueMultiplier* = 0.11. Estos valores se corresponden con la sensibilidad del ojo a cada color. Dado que el ojo humano es más sensible al verde, se le da mayor peso a la componente del verde en el valor del gris que las otras dos.

Este objeto *ColorTransform* es aplicado a toda la imagen contenida en el *BitmapData,* obteniendo una imagen en la que cada componente de color ha sufrido la transformación indicada, pero no una imagen en escala de grises. El siguiente paso es calcular el valor del gris de cada *pixel*, para lo cual hay que sumar los valores de cada una de las componentes.

Para hacer ésto es necesario almacenar en un *Array* los valores de cada *pixel* para lo cual se utiliza el método *getPixels*() del objeto *BitmapData*. Este método devuelve el *Array* necesario. Como en *ActionScript* cada *pixel* está codificado en ARGB, hay que tener en cuenta que la información relativa a cada uno de ellos ocupa cuatro posiciones en el *Array*.

Una vez se dispone del *Array* con los valores de todas las coordenadas, el siguiente paso es recorrerlo en bloques de cuatro en cuatro (lo que ocupa un *pixel*). Para

ello se emplea un bucle, el programa va sumando el valor de la segunda, tercera y cuarta posición del bloque para obtener el valor del gris de cada *pixel*. Cuando se obtiene el valor del gris, se le asigna a la segunda, tercera y cuarta posición del bloque de modo que todas las componentes de color tengan el mismo valor (dentro de cada *pixel*). Este proceso se repite para todos los *pixeles* de la imagen.

Una vez finalizado se pasan los nuevos valores del *Array* al objeto *BitmapData* y con esto ya se obtiene la imagen en escala de grises.

El último paso del proceso de transformación es pasar la imagen en escala de grises a blanco y negro. Para ello se hace uso de una función [13] (obtenida de internet) que se basa en un algoritmo de binarización por umbral adaptativo. Obtiene muy buenos resultados en diferentes condiciones de iluminación. La única modificación aplicada sobre dicho algoritmo consiste en cambiar la salida del mismo. El algoritmo sacaba los valores por debajo del umbral como negro y los que estaban por encima como blanco. Debido a que la herramienta *eyegrade* utiliza las imágenes en negativo se tuvo que transformar el algoritmo para que saque los valores por debajo del umbral como blanco y los que están por encima como negro.

5.5.- Binarización de la imagen.

La función de esta parte es intentar reducir en todo lo posible el tamaño de los datos a enviar al servidor, para un mejor aprovechamiento del ancho de banda. Como la información que guarda cada *pixel* es simplemente si está en blanco o en negro, ésta se puede codificar con un solo *bit*. Por lo tanto hay que transformar la información de la imagen almacenada en el *BitmapData* y guardarla en un *ByteArray* donde cada uno de los *bits* representen el valor de cada *pixel*. Cada posición del *ByteArray* almacena un *byte*, luego en cada una hay que almacenar el valor de 8 *pixeles*. El primer *byte* va a almacenar los 8 *pixeles* situados más a la izquierda de la línea superior de la imagen, de estos 8 *pixeles* el que está a la izquierda del todo es el más significativo dentro del byte. El resto de *bytes* se corresponderían con recorrer la imagen línea a línea, de arriba a abajo, y de izquierda a derecha dentro de cada línea.

La forma de conseguir esta binarización es la que se detalla en la siguiente función:

Función hacerByte

El primer paso es cargar los valores de cada *pixel* en un *Array*, para lo cual se vuelve a usar el método *getPixels*() del objeto *BitmapData* que contiene la imagen en blanco y negro. Después se crean tres variables nuevas: una para ir guardando el valor que debe contener el *byte*, la segunda para utilizar como índice del *ByteArray,* que será la salida de esta función, y la última para ir recorriendo el *Array* con los datos mediante un bucle *While*.

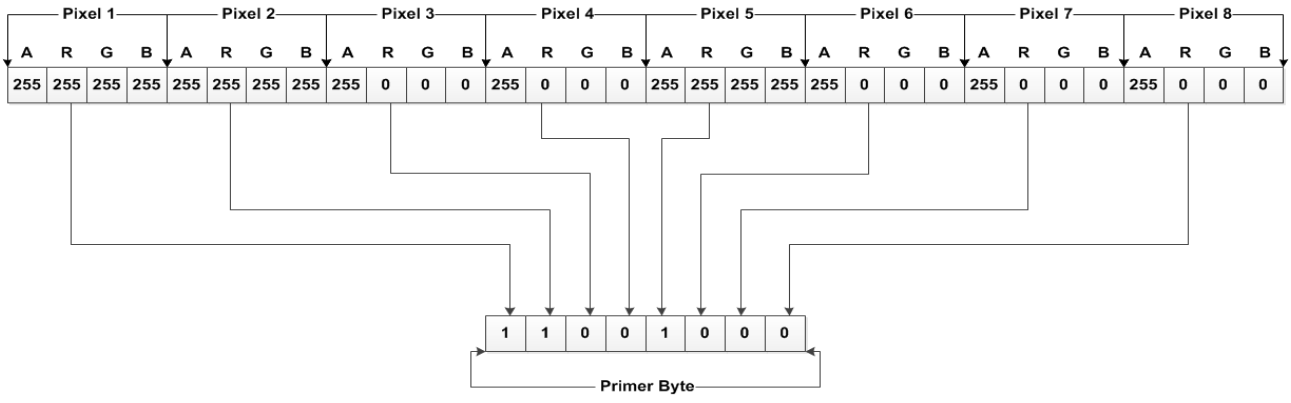
Dentro del bucle que irá leyendo los valores del *Array* se crea otro que recorra los números del siete al cero (que son las potencias de dos que hay dentro de un byte). En el caso de que la información del *pixel* leída del *Array* sea igual a 255 se guarda en la variable *byte* la información que contenía más el valor de la potencia de dos pertinente. Es decir para el primer *pixel* del *byte* se sumaría al valor de *byte* 128, para el segundo *pixel* del grupo, en caso de estar a uno también, se sumaría 64, etc..

En el caso de que algún *pixel* esté a cero no se sumaria nada a la variable *byte* y de este modo se completa el valor del *byte* a partir de la información contenida en 8 *pixeles*.

El bucle que recorre las posiciones del *Array* va dando saltos de cuatro en cuatro para así poder ir cogiendo un valor de cada *pixel*. Ya que cada pixel ocupa cuatro posiciones dentro del *Array* donde están todos los datos de la imagen.

Lo último es ir cargando en el *ByteArray* el valor del *byte* obtenido en cada grupo de 8 pixeles. Con esto se obtiene la información relevante de toda la imagen binarizada para un mejor aprovechamiento del ancho de banda.

En el siguiente esquema se detalla cómo se asignan los valores de los 8 primeros *pixeles* al primer *byte* del *Array*.



5.6.- Envío de las imágenes al servidor.

Para enviar los datos de la imagen al servidor se utiliza una petición HTTP con el método POST enviada al recurso */process* del mismo. En el cuerpo de la petición va encapsulada la variable *ByteArray* con los datos en binario. Para que el servidor entienda los datos hay que darle una cabecera indicando que el *Content-Type* es *application/x-eyegrade-bitmap*. Esto se hace configurando una nueva variable del tipo *URLRequest.* Ésta se pasa como parámetro de entrada en la creación de una variable del tipo *URLLoader*, que es cuando realmente se envían los datos al servidor.

Esta última tiene asociados dos escuchadores de eventos: uno para controlar cuando se recibe la respuesta del servidor y otro para ver que no haya ningún problema en la comunicación con el servidor.

5.7.- Análisis del XML devuelto por el servidor.

Si la petición ha llegado correctamente al servidor, éste responde a dicha petición con un documento XML. Hay dos tipos de documentos, véase el apéndice 3: uno que indica que no se ha podido realizar la corrección y otro que indica que la corrección se ha realizado correctamente. En este último caso, dicho documento XML también contiene todos los datos necesarios para poder pintar los símbolos de la corrección sobre la imagen que está viendo el usuario.

Cuando salta el evento de que se ha completado la comunicación entre cliente y servidor, el programa activa una función que es la encargada de obtener los datos de la respuesta y almacenarlos en una variable del tipo XML. El funcionamiento de esta función es el siguiente:

Lo primero que hace es leer el campo "*ok"*, véase el apéndice 3, del documento XML. En caso de que este campo contenga un valor *false*, indica que el servidor no ha podido corregir la imagen enviada. El programa vuelve a llamar a la función que se encarga de volver a iniciar el proceso de captura de una nueva imagen.

En caso de que dicho campo contenga un valor *true* quiere decir que el servidor ha sido capaz de realizar la corrección y que ha enviado los datos necesarios dentro del documento XML.

Para poder acceder a los distintos campos incluidos dentro del documento XML hay que partir de una variable del tipo XML. Dicha variable se crea cuando salta el evento que indica que se ha completado la comunicación entre el programa y el servidor. Esta variable se llama "datos" y se puede acceder al campo "*ok"* de la misma mediante la expresión "datos.*ok*". Por lo tanto cada campo dentro del XML es accesible desde la variable entrando en cada nivel mediante un punto. Es como si fuera un sistema de carpetas pero sustituyendo la barra “/” por el punto.

En caso de querer acceder a un atributo de un campo en concreto, hay que ir con el sistema indicado hasta el nivel donde se encuentre el campo requerido. Después hay dos opciones: se puede utilizar la propiedad *attribute* asociada a todo objeto XML, opción que sólo permite obtener los datos del atributo, o se puede poner una arroba “@” antes del nombre del atributo, opción que permite tanto obtener los datos como modificarlos en la variable XML.

Para obtener los datos del documento XML, *ActionScript* dispone del bucle *for each* que es muy útil para este tipo de documentos ya que permite dividirlo en bloques más pequeños. De este modo se pueden ir recorriendo todos los campos con el mismo nombre y llamando a las funciones de dibujo que se vayan necesitando.

5.8.- Pintado de una capa sobre la imagen capturada.

Este módulo del programa se compone de tres funciones. Cada una de ellas se encarga de presentarle al usuario un tipo diferente de información. Para que el usuario pueda ver esta información hay que crear previamente una capa de dibujo sobre la cual se van a ir colocando los distintos símbolos. Dicha capa es simplemente un objeto del tipo *Sprite*, que se coloca sobre la imagen del examen. En caso de no añadir ningún símbolo a esta capa, la misma es completamente transparente para el usuario.

Las tres funciones de este módulo son:

Función pintarPunto

Esta función recibe como parámetros de entrada las coordenadas X e Y donde se debe dibujar el punto. Con la propiedad *graphics* del objeto *Sprite* y haciendo uso de su

método *drawCircle*(), se pinta un circulo de color azul con un radio de tres *pixeles* en la posición indicada.

Este símbolo le indica al usuario que la celda donde está colocado es donde está la respuesta correcta. Además también le indica que el alumno no ha contestado bien la pregunta (ha fallado, la ha dejado sin responder o su respuesta es dudosa). Ya que en caso de que coincida la respuesta correcta con la del alumno este símbolo no se pinta sobre la imagen.

Función pintarCirculo

Esta función recibe como parámetros de entrada las coordenadas X e Y, el radio que debe tener el círculo en la pantalla y el color. El radio se obtiene de dividir entre 5 el tamaño de la diagonal de la celda sobre la cual hay que dibujar este símbolo. El color varía en función de si la respuesta es correcta (círculo verde) o no (círculo rojo). Sigue el mismo proceso que en la función anterior llamando al método *drawCircle*() pero en este caso simplemente se pinta el contorno del mismo.

Este símbolo le indica al usuario donde se ha localizado la respuesta del alumno por parte del servidor.

Función escribirDatos

Esta función es la que se encarga de poner los datos, leídos del documento XML, que envía el servidor como respuesta. Esta información se coloca en unas etiquetas que están situadas justo debajo de la imagen corregida. En estas etiquetas el programa muestra el nombre del alumno, su NIA, el número de aciertos, fallos, respuestas dudosas y respuestas en blanco.

5.9.- Detección de las celdas de la imagen donde el usuario pincha.

Este módulo es el encargado de posibilitar al usuario modificar la corrección efectuada por el servidor, por si se da el caso de que haya alguna pregunta con la que el usuario no esté de acuerdo con la corrección.

Para darle esta posibilidad al usuario es necesario asociar un escuchador al escenario general del programa. Dicho escuchador se encarga de capturar los eventos

que se lanzan cuando el usuario pincha sobre el escenario del programa. Teniendo en cuenta que solo se quiere que el usuario tenga la posibilidad de acceder a éste modulo cuando haya una imagen corregida, hay que comprobar primero que sea así y que el usuario ha pinchado sobre la imagen, ya que en caso contrario no debe hacer nada.

Cuando el usuario haya pinchado sobre una imagen corregida y en la zona donde están los símbolos de la corrección, el programa debe comprobar si el usuario ha pinchado encima de una casilla. En ese caso, modifica el XML de forma que si esa es la casilla donde estaba la respuesta del usuario la quita y en caso de que no coincida con la respuesta del usuario marca esa casilla como la respuesta en el documento.

Una vez hecha esta modificación llama al modulo que lee el XML para volver a pintar los símbolos sobre la imagen para que se actualice la corrección.

5.10.- Carga de ficheros locales en la aplicación.

En el caso de cargar una sesión preexistente en el disco local del usuario la función invocada al dar al botón es la siguiente:

Función cargarSesion.

Su misión es cargar los datos existentes desde un archivo que el usuario tenga guardado en su disco local. Para ello se utiliza un objeto *FileReference* al que se le añaden dos escuchadores. Uno para que lance un evento cuando se seleccione un archivo y el otro para cuando se complete la carga del mismo.

El método *browse*() de dicho objeto se utiliza para abrir un cuadro de diálogo donde el usuario puede seleccionar el archivo.

Función sesionSeleccionada.

Se activa cuando salta el evento *select* del objeto *FileReference* mencionado en la función anterior. Lo que hace es cargar el archivo seleccionado para poder acceder a los datos del mismo.

Función sesionCargada.

Se invoca cuando se ha completado la carga de los datos del archivo asociado al objeto *FileReference.* Una vez están disponibles los datos, lo que hace la función es convertirlos a tipo *String* y pasárselos a la variable que se había creado antes para ir almacenando los resultados de cada corrección.

5.11.- Guardado de ficheros en la máquina local.

Ésta es la función que se encarga de darle al usuario la posibilidad de almacenar el resumen de las correcciones en un archivo de texto en su equipo local.

Función guardarSesion

Lo primero que mira es si hay o no datos que almacenar en la variable local antes de guardar el resumen. Esto se produce cuando el usuario ha corregido el último examen y le da a cerrar sesión, de modo que los datos de esa última corrección no se han guardado aún en la variable auxiliar. Por lo tanto en caso de que sea este el caso, guarda esos datos en la variable.

Después comprueba si hay o no datos nuevos para ser guardados. El usuario ha podido abrir una sesión existente y no haber corregido ningún examen adicional, en cuyo caso no tiene sentido que el programa permita al usuario guardar nada. En este caso no hay información nueva que almacenar y la que ha sido cargada previamente ya está guardada en el disco local del usuario.

En el caso de que el usuario haya corregido algún examen el programa abre un cuadro de diálogo que le permite al usuario elegir donde quiere guardar el resumen de la corrección. Este cuadro le indica que la extensión que debería utilizar es ".txt" pero si el usuario quiere utilizar otra por el motivo que sea, el programa se lo permite. Hay que tener en cuenta que los datos que se van a guardar son del tipo *String,* por lo tanto el archivo con el resumen solo tiene sentido que sea de tipo texto, para que el usuario pueda después modificarlo o visualizarlo con cualquier editor de texto del que disponga en su equipo.

5.12.- Cerrado de la sesión entre el cliente y el servidor.

Esta función se encarga de decirle al servidor que el programa desea cerrar la comunicación entre ambos, debido a que el usuario ha decidido no seguir corrigiendo más exámenes para esa sesión.

Función cerrarSesion

El programa crea una nueva petición HTTP pero esta vez se envía con el método GET al recurso */close* del servidor.

La forma de crear esta petición es similar a las explicadas anteriormente para el resto de envíos al servidor. La única diferencia es que esta vez al objeto *URLRequest* hay que indicarle el método GET en lugar del POST. No lleva ningún tipo de *Content-Type* ya que en esta petición no va ningún tipo de datos encapsulados.

5.13.- Creación de mensajes *PopUp*.

Función mensaje.

Esta función recibe como entrada el texto que se mostrará al usuario por pantalla. Ésta se utiliza para dar información al usuario de diferentes incidencias a lo largo de la ejecución del programa.

Dentro de la extensa API de la que dispone *ActionScript* no hay ninguna forma de lanzar ventanas emergentes al usuario. Sin embargo sí que tiene la posibilidad de poder utilizar algunas instrucciones de código en *JavaScript*. Para esta función en concreto con una simple instrucción en *JavaScript* se consigue el efecto buscado.

Así que para poder lanzar mensajes emergentes se ha creado una nueva variable del tipo *URLRequest* con la siguiente dirección "javascript:alert('" + texto + "')". Donde texto hace referencia a la entrada que recibe la función. Una vez creada la nueva variable solo queda utilizar el método navigateToURL(), indicando la *URLRequest* creada anteriormente para que aparezca la ventana emergente con el texto elegido.

Capítulo 6.- Pruebas.

Este capítulo explica las diferentes pruebas que se han ido realizando durante el desarrollo de este proyecto. Para ello se va a utilizar el orden relatado en el diagrama de Gantt expuesto en el anexo, de modo que siga un orden cronológico.

6.1.- Pruebas iniciales.

Las primeras pruebas que se realizaron fueron para comprobar que se podía capturar la cámara *web* con Flash y mostrarle dicha captura al usuario. Esta parte no presentó ningún problema y se dio por buena.

6.2.- Pruebas para detener la imagen.

Durante la realización de estas pruebas se han utilizado diferentes métodos para intentar detener la captura de imágenes de la cámara *web* y mostrarle al usuario una imagen fija. La creación de un objeto *Bitmap* resulta ser la forma óptima de conseguirlo.

En un principio se intentó detener la reproducción en el objeto de video que muestra las imágenes capturadas por la cámara, pero no era la solución al problema ya que después no se podían utilizar los datos contenidos en dicha imagen. También se intentó parar la captura de la propia cámara pero no se consiguió el efecto buscado. Por lo que finalmente se optó por utilizar un objeto *Bitmap* para detener la captura y un objeto *BitmapData* para obtener los datos de la imagen.

6.3.- Pruebas para pasar la imagen a gris.

Este bloque de pruebas se centró en, una vez obtenida la imagen contenida en el objeto *Bitmap*, pasar la imagen que estaba en color a escala de grises. En los primeros intentos la imagen salía en un predominante tono verde. El problema estaba en que al hacer la ponderación de las componentes RGB el verde tenía más peso que el resto y no se habían sumado todos los valores después de ser ponderados. Una vez subsanado este error se obtuvo la imagen correctamente.

6.4.- Pruebas para pasar la imagen a blanco y negro.

Buscando la forma óptima de pasar la imagen a blanco y negro se realizaron muchas pruebas con distintas formas de calcular el umbral que debería ser aplicado a la imagen en gris. La primera de ellas consistió en calcular el nivel medio de gris de toda la imagen y a partir de ese valor clasificar cada *pixel* como blanco o negro, pero ésto no dio buen resultado. En la siguiente prueba se parceló la imagen en subgrupos más pequeños de *pixeles*, pero tampoco se obtuvo un resultado bueno. Por último se utilizó un algoritmo de umbral adaptativo [13], el cual da muy buenos resultados en diferentes condiciones de iluminación.

6.5.- Pruebas de binarización de la imagen.

Esta parte consistió en ver cuál era la mejor forma de empaquetar los datos de la imagen de modo que cada *pixel* ocupase solo un *bit*. Dado que cada uno de ellos solo guarda la información de si es negro o blanco, con un *bit* es más que suficiente para codificar su información.

Se optó por agrupar los *pixeles* en grupos de ocho para poder formar un *byte,* que es la unidad mínima de información que se puede manejar con *ActionScript* 3.0. Dado que cada línea tiene 640 *pixeles* (que es múltiplo de 8) todos los representados en un mismo *byte* pertenecen a la misma línea.

Por último solo quedaba definir cómo se iban a ordenar dichos *pixeles* y se optó por hacer que el *bit* más significativo del *byte* se correspondiera con el *pixel* situado más a la izquierda del grupo de ocho. Y por lo tanto el que está más a la derecha con el *bit* de menos peso del *byte*.

6.6.- Pruebas envío de imágenes al servidor.

Este bloque de pruebas se llevó a cabo gracias la herramienta *Wireshark*, lo que permitió ir capturando el tráfico que se generaba entre la aplicación y el servidor. Esta parte fue fundamental durante el desarrollo de las mismas, ya que utilizando simplemente las trazas que lanza la aplicación no era posible detectar el origen de los errores que fueron surgiendo.

Gracias a dicha herramienta se descubrió un error en la cabecera de las peticiones HTTP que se enviaban al servidor, en el cual no coincidía el *Content-Type* con el tipo de datos que se enviaban encapsulados en el cuerpo del mensaje.

Una vez subsanado dicho error el envío de imágenes funcionaba de una forma correcta y se dieron por buenas las pruebas.

6.7.- Pruebas de lectura de un documento XML.

Estas pruebas se han centrado en ver qué herramientas ofrece el *ActionScript 3.0* para el manejo de documentos XML, para saber cómo se puede recorrer un documento extrayendo información del mismo. Durante las mismas el principal problema que se encontró fue a la hora de poder cambiar datos de un atributo perteneciente a una etiqueta de dicho documento.

Para obtener la información de un atributo, el objeto XML dispone de un método llamado *attribute()*. Éste devuelve la información almacenada en dicho atributo, pero no permite modificarla. Después de diversas pruebas se llegó a la solución, la cual consiste en utilizar una @ como prefijo del nombre del atributo para que se puedan modificar los datos.

6.8.- Pruebas de carga de archivos locales.

En un primer momento se consiguió realizar la carga de archivos para poder enviarlos al servidor sin ningún tipo de problemas. Pero a la hora de cargar datos desde un archivo para manejarlos dentro de la aplicación, no resultó tan sencillo.

Para conseguir manejar los datos dentro de la aplicación es necesario utilizar diferentes eventos, realizando las acciones necesarias en el momento preciso. Es decir, no se puede cargar el archivo antes de que esté seleccionado, no se pueden pasar los datos a una variable antes de que se haya completado la carga, etc.

Finalmente esta solución también tuvo que ser adoptada para cargar los datos que se enviaban al servidor. Aunque con el primer intento se enviaban bien (al menos desde el ejecutable de la aplicación en local) resultó no ser portable a diferentes navegadores,

solo funcionaba correctamente con *Internet Explorer*. Esta parte se explicará con más detalle en el siguiente grupo de pruebas.

6.9.- Pruebas de envío de archivos al servidor.

Las primeras pruebas se realizaron utilizando el método *upload()* de un objeto *FileReference*, el cual subía fácilmente el archivo al servidor, pero había un problema con las cabeceras del mensaje que enviaba el archivo. Al utilizar dicho método *ActionScript* no se permite modificar el *Content-Type* del mismo y hubo que realizar variaciones en el servidor para que éste aceptase un nuevo tipo de datos.

Cuando este problema fue subsanado se comprobó que con este método tampoco era posible mantener el identificador de la sesión, siempre que se utilizase un navegador diferente al *Internet Explorer*. Por lo que hubo que modificar la forma de enviar los datos, para hacerlo compatible con más navegadores y por ende, con más sistemas operativos.

Finalmente se adoptó la forma de cargar los archivos descrita en el punto anterior y el modo de enviar los datos es muy similar al que utilizaba la aplicación para enviar las imágenes.

6.10.- Pruebas para modificar la corrección enviada por el servidor.

En esta parte el principal problema que apareció fue a la hora de borrar los símbolos que estaban pintados sobre la imagen, sacados de la corrección enviada por el servidor. El problema es que *ActionScript* no permite borrar los símbolos de uno en uno, sino que borra una capa entera. Por lo tanto, la solución adoptada fue limpiar la capa por completo y volver a pintarla cada vez que el usuario modificaba algo de la corrección.

La forma de conseguir ésto es mediante la modificación del documento XML, que es el que le indica a la aplicación dónde debe colocar cada símbolo. Por lo tanto cada vez que el usuario realiza alguna modificación sobre la corrección, ésta se aplica sobre el documento y después se procede a limpiar la corrección anterior y volver a pintar la nueva.

6.11.- Pruebas de uso de cadenas de texto con formato.

Esta parte no presentó ningún tipo de problemas, básicamente consistió en buscar el mejor formato para almacenar los datos en el disco local del usuario.

Teniendo en cuenta que esta aplicación funciona en colaboración con la herramienta *eyegrade* se optó por utilizar el mismo formato de datos que vienen indicados en dicha herramienta.

6.12.- Pruebas de guardado de archivos en el disco local del usuario.

Estas pruebas salieron correctamente pudiendo acoplar este módulo a los anteriores sin ningún contratiempo.

6.13.- Pruebas finales.

Una vez completadas todas las pruebas parciales (de cada módulo) se procedió a comprobar el correcto funcionamiento del sistema general.

El funcionamiento de la aplicación era correcto y el flujo de datos era el programado previamente, por lo que los únicos ajustes que hubo que realizar respondían más a un cuestión de estilo que a un problema en la ejecución del sistema.

Los últimos ajustes realizados consistieron en cambiar el color de los símbolos que identifican la respuesta del usuario. En la primera versión se usaba el mismo color si la respuesta era correcta o no, mientras que una vez revisado se empezó a usar el verde si la respuesta del usuario era correcta, el rojo en caso contrario y el azul para los puntos que identifican la solución.

Además también se incluyó un botón que le permite al usuario cancelar el envío de imágenes al servidor, por si desea cerrar la sesión o simplemente volver a colocar la cámara en una posición diferente.

Capitulo 7.- Conclusiones y trabajos futuros.

7.1.- Conclusiones del proyecto.

Una vez analizados todos los objetivos planteados al comienzo de este proyecto de fin de carrera y el resultado final del mismo, se ha conseguido dotar al programa de todas las funcionalidades necesarias para cubrir las necesidades del usuario.

La principal dificultad encontrada en la realización de este proyecto ha sido que no tenía ningún tipo de conocimiento previo de *Flash* ni de *ActionScript*. Sin embargo, se ha encontrado que *Flash* dispone de una documentación útil para programadores, en especial la documentación de la API que provee para *ActionScript*.

Mediante el uso de Flash, se ha conseguido que el programa sea portable, esto es, compatible con diversos navegadores y sistemas operativos.

Durante las diferentes pruebas llevadas a cabo se ha obtenido un tiempo de respuesta del programa aceptable para poder utilizar la aplicación de forma continuada. El programa responde mejor cuantos más exámenes se van corrigiendo, debido a que el servidor va autoajustando los parámetros de detección de líneas, es necesario enviar menos capturas al servidor para corregir un examen si se han realizado correcciones previas. Por lo tanto no es especialmente útil para corregir un solo examen pero sí cuando se corrigen varios.

Finalmente *Flash* ha demostrado, al menos en la realización de este Proyecto, ser una herramienta realmente efectiva para el desarrollo de aplicaciones *web* de un modo sencillo. Permite acceder a todos los elementos multimedia conectados al equipo del desarrollador y genera aplicaciones con un tamaño muy reducido para poder ser embebidas en una *web*.

En cuanto a los puntos fuertes del programa cabe destacar la capacidad para poder acceder a todas las cámaras disponibles en el equipo del usuario. La portabilidad del sistema y el hecho de que no es necesario instalarlo en el equipo del usuario. La posibilidad de utilizar archivos locales para cargar y almacenar datos, ya que de este modo el usuario puede modificar y consultar dichos datos con cualquier editor de texto.

Mientras que en los aspectos a mejorar, entraría el aprovechamiento del ancho de banda, ya que aplicando algún tipo de algoritmo de compresión sin pérdidas se podría disminuir aún más el tamaño de las imágenes enviadas. También se podría añadir una nueva vista donde se mostrasen los datos de las correcciones realizadas hasta el momento para que el usuario compruebe que exámenes ha corregido.

7.2.- Trabajos futuros.

El motivo principal de este proyecto es el envío de imágenes desde un equipo local a un servidor para su posterior procesado. Por lo tanto con unas simples modificaciones en esta aplicación, se podría utilizar para el procesado de cualquier tipo de imágenes por parte del servidor.

Algunas aplicaciones que podrían ser cubiertas por este proyecto (realizando alguna modificación y preparando el servidor para cada caso) son:

Envío de imágenes del usuario para su reconocimiento facial. Implementación de un sistema de identificación mediante el análisis de su pupila. Envío de imágenes para su clasificación en diferentes categorías, para usarse como reconocedor de objetos o entornos. Para la creación de avatares, con las características físicas del usuario, en juegos *online*, de modo que el usuario envíe una imagen suya capturada por la cámara *web* para la creación del personaje que va a manejar.

En definitiva, podría ser útil para cualquier aplicación que necesite captar imágenes en un entorno local pero que sean posteriormente tratadas por un servidor, tanto a nivel académico como profesional.

Otra posibles ampliaciones del Proyecto serían:

Cuando se estandarice el acceso mediante HTML 5 a la cámara *web*, y los navegadores le den soporte, reprogramación del sistema sin Flash, mediante JavaScript y las APIs de HTML 5.

Aplicación *web* de gestión integral del proceso de corrección, calificaciones, etc. Se trataría de tener una aplicación *web* con base de datos que guarde listados de alumnos, permita crear exámenes, corregirlos, gestionar la hoja de notas, compartir exámenes entre profesores, analizar estadísticas de resultados, etc., todo ello en el navegador.

Aplicación web enfocada a la resolución de circuitos eléctricos, el usuario tendría que enfocar un dibujo de un circuito y el programa se encargaría de calcular las corrientes en cada una de las ramas, voltajes, etc. Se podría utilizar para que los alumnos puedan comprobar la resolución de ejercicios propuestos en clase o incluso otros circuitos diseñados por ellos mismos.

Presupuesto.

Para la realización de este proyecto se han empleado los siguientes recursos:

- ✓ Un ordenador.
- ✓ Un programador.
- ✓ Una cámara *web*.
- ✓ Licencia de Adobe *Flash*.

El coste de los medios técnicos se desglosa de la siguiente manera:

Recurso	**Coste**	**Vida útil**	**Amortización**	**Total**
Ordenador	650€	5 años	8%	**52€**
Cámara *web*	10€		100%	**10€**
Licencia *Flash*	349€		100%	**349€**
				411€

El coste del programador, con una jornada de 8 horas al día y de lunes a viernes, es de:

Precio / Hora	**Mes**	**Días / Mes**	**Horas / Mes**	**Total**
30€	Junio	22 Días	176 Horas	**5280€**
30€	Julio	21 Días	168 Horas	**5040€**
30€	Agosto	23 Días	184 Horas	**5520€**
30€	Septiembre	22 Días	176 Horas	**5280€**
30€	Octubre	12 Días	96 Horas	**2880€**
				24000€

Por lo tanto el coste total del Proyecto asciende a **24411€**.

Apéndices.

1.- Archivo de configuración.

La estructura del archivo de configuración viene marcada por la aplicación *Eyegrade*, un ejemplo de este archivo seria:

```
[exam]

num-models: 8

dimensions: 4,10;4,10

id-num-digits: 9



[solutions]

model-A: 3/2/1/2/1/4/1/4/2/2/2/1/2/4/3/2/4/3/3/3

model-B: 3/4/4/2/4/4/3/3/2/1/4/2/1/1/4/1/3/1/1/4

model-C: 1/3/2/4/4/2/1/4/1/2/4/3/4/2/4/3/3/1/4/3

model-D: 3/2/2/1/2/2/4/1/4/3/3/1/1/4/4/3/2/2/4/1

model-E: 1/4/1/1/3/2/3/1/3/3/1/4/4/3/4/4/4/1/1/1

model-F: 2/2/3/4/4/3/3/2/4/1/3/2/3/1/2/3/2/3/4/3

model-G: 4/2/1/2/2/3/1/2/3/1/1/3/2/1/1/1/4/4/4/1

model-H: 1/1/1/3/2/2/4/4/4/2/1/1/1/4/1/2/2/4/4/2
```

En él se indican el número de preguntas, la geometría del examen, las soluciones, ect... Se aconseja que este archivo tenga una extensión .txt u otra de texto.

2.- Archivo con la lista de alumnos.

La estructura del archivo que contiene la lista de alumnos es muy sencilla, simplemente debe contener el NIA del alumno, un tabulador (utilizado como separador) y el nombre del alumno. Un ejemplo de lista seria:

```
100099999 John Doe

100099998 Jane Doe

100071430 Diego Serrano Toca
```

Se aconseja que este archivo tenga una extensión de tipo texto (.txt u otras).

3.- Respuestas del servidor al envío de imágenes.

En este apéndice se van a comentar los dos tipos de documentos que envía el servidor como respuesta al envío de imágenes.

3.1.- Corrección realizada.

En el caso de que el servidor haya podido realizar la corrección, envía un documento XML con el siguiente esquema:

```
<output>
    <ok>true</ok>
    <model>A</model>
    <student id="100071430">Diego Serrano Toca</student>
    <result>
     <good>7</good>
     <bad>10</bad>
     <blank>3</blank>
     <unclear>0</unclear>
    </result>
    <answers>
     <answer num="1" correct="false" answered="4" solution="3" />
     <answer num="2" correct="false" answered="3" solution="2" />
     <answer num="3" correct="true" answered="1" solution="1" />
     <answer num="4" correct="false" answered="0" solution="2" />
     <answer num="5" correct="false" answered="0" solution="1" />
```

```
 <answer num="6" correct="true" answered="4" solution="4" />
 <answer num="7" correct="false" answered="3" solution="1" />
 <answer num="8" correct="false" answered="2" solution="4" />
 <answer num="9" correct="false" answered="1" solution="2" />
 <answer num="10" correct="true" answered="2" solution="2" />
 <answer num="11" correct="false" answered="0" solution="2" />
 <answer num="12" correct="false" answered="4" solution="1" />
 <answer num="13" correct="true" answered="2" solution="2" />
 <answer num="14" correct="false" answered="1" solution="4" />
 <answer num="15" correct="false" answered="1" solution="3" />
 <answer num="16" correct="true" answered="2" solution="2" />
 <answer num="17" correct="true" answered="4" solution="4" />
 <answer num="18" correct="false" answered="1" solution="3" />
 <answer num="19" correct="false" answered="1" solution="3" />
 <answer num="20" correct="true" answered="3" solution="3" />
</answers>
<geometry>
 <question num="1">
      <cell choice="1" center_x="89" center_y="128"
      diagonal="55.2268050859" />
      <cell choice="2" center_x="137" center_y="128"
      diagonal="56.080299574" />
```

```
    <cell choice="3" center_x="187" center_y="128"
    diagonal="57.454329689" />

    <cell choice="4" center_x="237" center_y="129"
    diagonal="57.454329688" />

</question>

<question num="2">

    <cell choice="1" center_x="88" center_y="156"
    diagonal="56.6038867923" />

    <cell choice="2" center_x="137" center_y="157"
    diagonal="57.454329689" />

    <cell choice="3" center_x="186" center_y="158"
    diagonal="58.302643203" />

    <cell choice="4" center_x="236" center_y="158"
    diagonal="58.305189485" />

</question>

<question num="3">

    <cell choice="1" center_x="88" center_y="185"
    diagonal="55.7584074378" />

    <cell choice="2" center_x="136" center_y="186"
    diagonal="56.038867923" />

    <cell choice="3" center_x="186" center_y="187"
    diagonal="57.454296889" />

    <cell choice="4" center_x="236" center_y="188"
    diagonal="57.454329689" />

</question>
```

```
<question num="4">

     <cell choice="1" center_x="87" center_y="215"
     diagonal="55.4707129934" />

     <cell choice="2" center_x="135" center_y="216"
     diagonal="57.140178086" />

     <cell choice="3" center_x="185" center_y="217"
     diagonal="57.982756573" />

     <cell choice="4" center_x="235" center_y="218"
     diagonal="57.982750573" />

</question>

<question num="5">

     <cell choice="1" center_x="87" center_y="245"
     diagonal="56.302753041" />

     <cell choice="2" center_x="135" center_y="246"
     diagonal="57.98260573" />

     <cell choice="3" center_x="184" center_y="247"
     diagonal="58.830264323" />

     <cell choice="4" center_x="234" center_y="248"
     diagonal="58.830264203" />

</question>

<question num="6">

     <cell choice="1" center_x="86" center_y="274"
     diagonal="54.918120871" />

     <cell choice="2" center_x="134" center_y="275"
     diagonal="56.603867923" />
```

```
        <cell choice="3" center_x="184" center_y="276"
        diagonal="57.454326889" />

        <cell choice="4" center_x="234" center_y="277"
        diagonal="57.454326889" />

    </question>

    <question num="7">

        <cell choice="1" center_x="86" center_y="303"
        diagonal="55.7584074378" />

        <cell choice="2" center_x="134" center_y="304"
        diagonal="57.454329889" />

        <cell choice="3" center_x="183" center_y="306"
        diagonal="57.982756573" />

        <cell choice="4" center_x="233" center_y="307"
        diagonal="58.830264203" />

    </question>

    <question num="8">

        <cell choice="1" center_x="85" center_y="333"
        diagonal="56.0357029045" />

        <cell choice="2" center_x="133" center_y="334"
        diagonal="57.688820074" />

        <cell choice="3" center_x="182" center_y="335"
        diagonal="57.688820074" />

        <cell choice="4" center_x="232" center_y="337"
        diagonal="58.830264203" />

    </question>
```

```
<question num="9">

     <cell choice="1" center_x="84" center_y="363"
     diagonal="54.3783044973" />

     <cell choice="2" center_x="132" center_y="364"
     diagonal="56.603886923" />

     <cell choice="3" center_x="182" center_y="365"
     diagonal="57.454329889" />

     <cell choice="4" center_x="231" center_y="367"
     diagonal="58.830264203" />

</question>

<question num="10">

     <cell choice="1" center_x="84" center_y="391"
     diagonal="54.918120871" />

     <cell choice="2" center_x="132" center_y="393"
     diagonal="57.982760573" />

     <cell choice="3" center_x="181" center_y="394"
     diagonal="57.454329689" />

     <cell choice="4" center_x="231" center_y="396"
     diagonal="58.830264303" />

</question>

<question num="11">

     <cell choice="1" center_x="381" center_y="131"
     diagonal="58.830264203" />

     <cell choice="2" center_x="432" center_y="132"
     diagonal="58.830263203" />
```

```
        <cell choice="3" center_x="483" center_y="132"
        diagonal="58.830263203" />

        <cell choice="4" center_x="534" center_y="133"
        diagonal="58.830263203" />

    </question>

    <question num="12">

        <cell choice="1" center_x="381" center_y="160"
        diagonal="59.169248699" />

        <cell choice="2" center_x="432" center_y="161"
        diagonal="59.169248699" />

        <cell choice="3" center_x="483" center_y="162"
        diagonal="59.169248699" />

        <cell choice="4" center_x="534" center_y="162"
        diagonal="59.169248699" />

    </question>

    <question num="13">

        <cell choice="1" center_x="380" center_y="191"
        diagonal="58.830264203" />

        <cell choice="2" center_x="431" center_y="191"
        diagonal="59.363288186" />

        <cell choice="3" center_x="482" center_y="192"
        diagonal="59.363288186" />

        <cell choice="4" center_x="533" center_y="193"
        diagonal="59.363288186" />

    </question>
```

```
<question num="14">

     <cell choice="1" center_x="379" center_y="221"
     diagonal="58.830263203" />

     <cell choice="2" center_x="430" center_y="222"
     diagonal="58.830263203" />

     <cell choice="3" center_x="481" center_y="223"
     diagonal="58.830264203" />

     <cell choice="4" center_x="532" center_y="224"
     diagonal="58.830263203" />

</question>

<question num="15">

     <cell choice="1" center_x="379" center_y="251"
     diagonal="59.682492455" />

     <cell choice="2" center_x="429" center_y="252"
     diagonal="58.830263203" />

     <cell choice="3" center_x="480" center_y="253"
     diagonal="58.830263203" />

     <cell choice="4" center_x="532" center_y="254"
     diagonal="59.682493455" />

</question>

<question num="16">

     <cell choice="1" center_x="378" center_y="281"
     diagonal="58.830263203" />

     <cell choice="2" center_x="429" center_y="282"
     diagonal="58.830264203" />
```

```
    <cell choice="3" center_x="480" center_y="283"
    diagonal="59.682492455" />

    <cell choice="4" center_x="531" center_y="284"
    diagonal="60.207972894" />

</question>

<question num="17">

    <cell choice="1" center_x="378" center_y="310"
    diagonal="59.169248699" />

    <cell choice="2" center_x="428" center_y="312"
    diagonal="58.830263203" />

    <cell choice="3" center_x="479" center_y="313"
    diagonal="58.830264203" />

    <cell choice="4" center_x="531" center_y="314"
    diagonal="59.682493455" />

</question>

<question num="18">

    <cell choice="1" center_x="377" center_y="340"
    diagonal="59.363283186" />

    <cell choice="2" center_x="428" center_y="341"
    diagonal="58.523499536" />

    <cell choice="3" center_x="478" center_y="343"
    diagonal="58.830263203" />

    <cell choice="4" center_x="530" center_y="344"
    diagonal="60.207972894" />

</question>
```

```
    <question num="19">

        <cell choice="1" center_x="376" center_y="371"
        diagonal="58.830264203" />

        <cell choice="2" center_x="427" center_y="372"
        diagonal="57.982756573" />

        <cell choice="3" center_x="477" center_y="373"
        diagonal="59.363288186" />

        <cell choice="4" center_x="529" center_y="374"
        diagonal="60.207972894" />

    </question>

    <question num="20">

        <cell choice="1" center_x="376" center_y="400"
        diagonal="60.207972894" />

        <cell choice="2" center_x="426" center_y="402"
        diagonal="59.363288186" />

        <cell choice="3" center_x="477" center_y="403"
        diagonal="60.207972894" />

        <cell choice="4" center_x="528" center_y="405"
        diagonal="61.057350899" />

    </question>

   </geometry>

</output>
```

Este ejemplo refleja la respuesta del servidor a un examen del tipo A dado en el archivo de configuración expuesto en el apéndice 1. Dicho examen tiene 20 preguntas y 4 posibles opciones en cada una de ellas.

3.2.- Imagen no corregida.

En el caso de que no se haya podido corregir, por parte de la herramienta *Eyegrade*, la imagen enviada por la aplicación, el servidor envía la siguiente repuesta:

```
<output>

    <ok>false</ok>

<output>
```

Con esto es suficiente para que el programa envíe de forma automática una nueva imagen al servidor.

4.- Resumen de la corrección.

La estructura del fichero donde se almacena el resumen de las correcciones hechas durante la sesión tiene el siguiente formato:

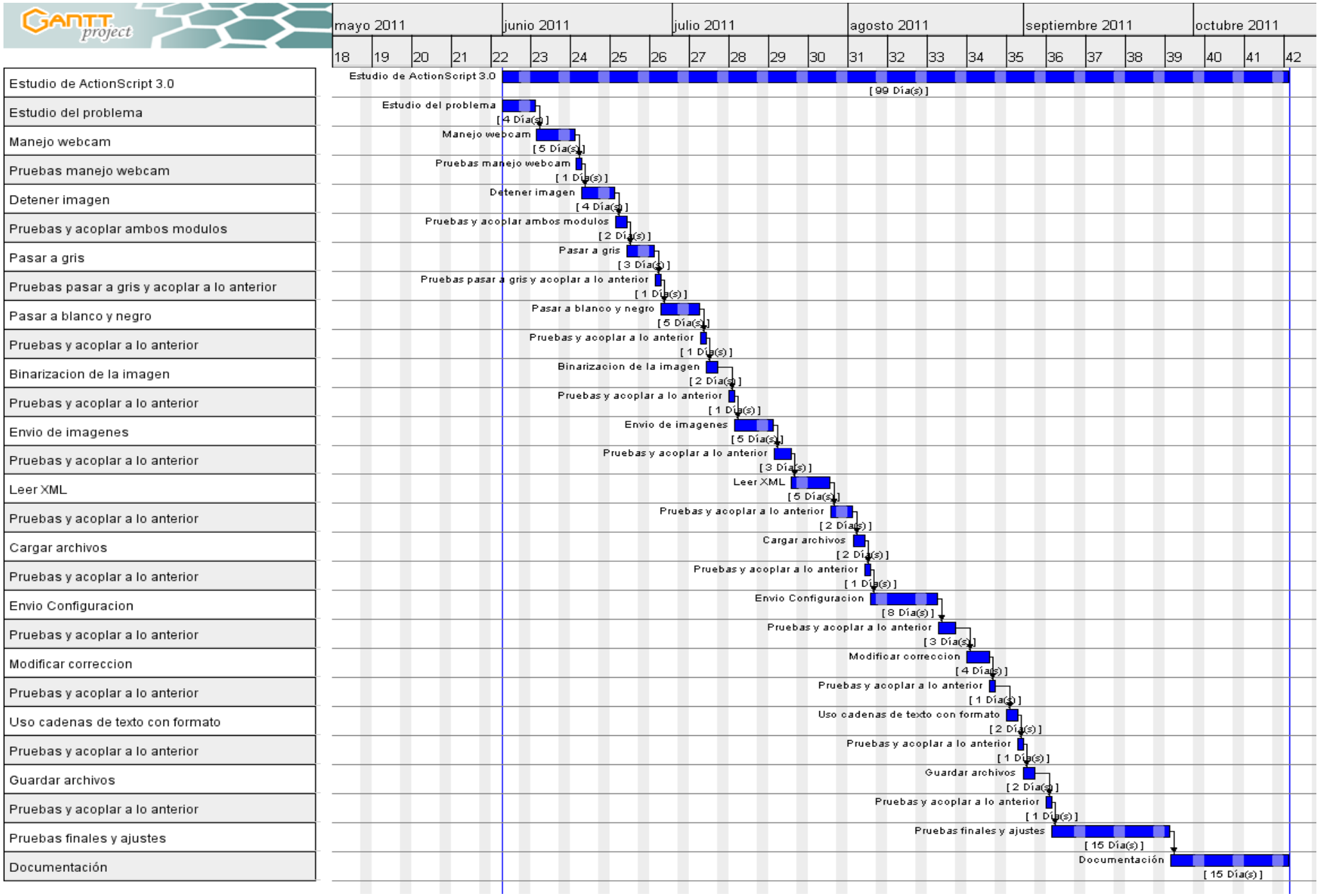
```
0  100999991  D  9  6  0  1/2/2/4/1/2/2/0/0/3/2/0/3/2/0/4/3/0/1/2

1  100999997  C  15  1  0  2/4/4/3/1/0/1/2/1/1/0/1/0/4/3/0/1/4/3/4

2  100800003  D  6  14  0  4/2/2/2/1/2/1/3/2/1/3/1/2/1/3/1/4/1/4/3

3  100777777  A  7  13  0  3/2/3/2/3/3/2/4/3/1/3/1/4/1/4/2/2/3/4/2
```

Este formato se ha extraído del indicado en el manual de la herramienta *Eyegrade*. El primer valor indica el numero de secuencia en la corrección, el segundo el NIA del alumno, el tercero el tipo de examen, el cuarto el número de aciertos, el quinto el número de fallos, el sexto el número de respuesta en blanco y dudosas y el último es la secuencia de las respuestas del alumno para ese examen. El separador utilizado para los campos es un tabulador.

Se recomienda guardar este archivo con un formato de tipo texto, es decir usar una extensión .txt o similar, para que se pueda consultar y modificar con cualquier editor de texto disponible en el equipo local.

5.- Diagrama de Gantt.

En este apéndice se expone un diagrama con la duración de cada una de las tareas llevadas a cabo durante la realización de este Proyecto. Además de una tabla donde vienen detalladas las fechas de inicio y fin de las mismas.


GANTT project
mayo 2011
junio 2011
julio 2011
agosto 2011
septiembre 2011
octubre 2011
18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
Estudio de ActionScript 3.0
Estudio del problema
Manejo webcam
Pruebas manejo webcam
Detener imagen
Pruebas y acoplar ambos modulos
Pasar a gris
Pruebas pasar a gris y acoplar a lo anterior
Pasar a blanco y negro
Pruebas y acoplar a lo anterior
Binarizacion de la imagen
Pruebas y acoplar a lo anterior
Envio de imagenes
Pruebas y acoplar a lo anterior
Leer XML
Pruebas y acoplar a lo anterior
Cargar archivos
Pruebas y acoplar a lo anterior
Envio Configuracion
Pruebas y acoplar a lo anterior
Modificar correccion
Pruebas y acoplar a lo anterior
Uso cadenas de texto con formato
Pruebas y acoplar a lo anterior
Guardar archivos
Pruebas y acoplar a lo anterior
Pruebas finales y ajustes
Documentación
[99 Día(s)]
[4 Día(s)]
[5 Día(s)]
[1 Día(s)]
[4 Día(s)]
[2 Día(s)]
[3 Día(s)]
[1 Día(s)]
[5 Día(s)]
[1 Día(s)]
[2 Día(s)]
[1 Día(s)]
[5 Día(s)]
[3 Día(s)]
[5 Día(s)]
[2 Día(s)]
[2 Día(s)]
[1 Día(s)]
[8 Día(s)]
[3 Día(s)]
[4 Día(s)]
[1 Día(s)]
[2 Día(s)]
[1 Día(s)]
[2 Día(s)]
[1 Día(s)]
[15 Día(s)]
[15 Día(s)]

Tarea	F. Inicio	F. Fin
Estudio *ActionScript* 3.0	01/06/2011	18/10/2011
Estudio del problema	01/06/2011	07/06/2011
Manejo cámara *web*	07/06/2011	14/06/2011
Pruebas manejo cámara web	14/06/2011	15/06/2011
Detener imagen	15/06/2011	21/06/2011
Pruebas y acoplar ambos	21/06/2011	23/06/2011
Pasar a gris	23/06/2011	28/06/2011
Pruebas pasar a gris y acoplar a lo anterior	28/06/2011	29/06/2011
Pasar a blanco y negro	29/06/2011	06/07/2011
Pruebas y acoplar a lo anterior	06/07/2011	07/07/2011
Binarización de la imagen	07/07/2011	09/07/2011
Pruebas y acoplar a lo anterior	11/07/2011	12/07/2011
Envío de imágenes	12/07/2011	19/07/2011
Pruebas y acoplar a lo anterior	19/07/2011	22/07/2011
Leer XML	22/07/2011	29/07/2011

Pruebas y acoplar a lo anterior	29/07/2011	02/08/2011
Cargar archivos	02/08/2011	04/08/2011
Pruebas y acoplar a lo anterior	04/08/2011	05/08/2011
Envío configuración	05/08/2011	17/08/2011
Pruebas y acoplar a lo anterior	17/08/2011	20/08/2011
Modificar corrección	22/08/2011	26/08/2011
Pruebas y acoplar a lo anterior	26/08/2011	27/08/2011
Uso cadenas de texto con formato	29/08/2011	31/08/2011
Pruebas y acoplar a lo anterior	31/08/2011	01/09/2011
Guardar archivos	01/09/2011	03/09/2011
Pruebas y acoplar a lo anterior	05/09/2011	06/09/2011
Pruebas finales y ajustes	06/09/2011	27/09/2011
Documentación	27/09/2011	18/10/2011

Referencias bibliográficas.

[1] UC3M, “Eyegrade User Manual”. Jesús Arias Fisteus. Disponible en línea: <http://www.it.uc3m.es/jaf/eyegrade/doc/user-manual/>, [acceso Octubre de 2011]

[2] Wikimedia Foundation, "Adobe Flash Professional". Wikipedia. Disponible en línea: <http://es.wikipedia.org/wiki/Adobe_Flash_Professional>, [acceso Octubre 2011]

[3] Adobe Systems Incorporated, “Programming ActionScript 3.0”. Adobe. Disponible en línea: <http://livedocs.adobe.com/flash/9.0_es/main/flash_as3_programming.pdf>, [acceso Octubre 2011]

[4] Adobe Systems Incorporated, “Ref. del lenguaje y componentes ActionScript 3.0”. Disponible en línea: <http://livedocs.adobe.com/flash/9.0_es/ActionScriptLangRefV3/>, [acceso Octubre 2011]

[5] Wikimedia Foundation, “Microsoft Silverlight”. Wikipedia. Disponible en línea: <http://es.wikipedia.org/wiki/Microsoft_Silverlight>, [acceso Octubre 2011]

[6] Sams, “Silverlight™ 4 Unleashed”. Laurent Bugnion.

[7] Blog de WordPress.com, “Capturar Video y Tomar Fotos – Silverlight”. Martín Reina. Disponible en línea: <http://escarbandocodigo.wordpress.com/2010/10/07/capturar-video-y-tomar-fotos-silverlight/>, [acceso Noviembre 2011]

[8] Geeks | Blogs. “Silverlight y los Sockets”. Omar del Valle. Disponible en línea: <http://geeks.ms/blogs/omarvr/archive/2010/10/27/silverlight-y-los-sockets-estableciendo-conexiones-con-el-servidor.aspx>, [acceso Noviembre 2011]

[9] Wikimedia Foundation, “HTML5”. Wikipedia. Disponible en línea: <http://es.wikipedia.org/wiki/HTML_5>, [acceso Octubre 2011]

[10] El Mundo.es | Blogs. “HTML5: Grandes mitos al descubierto”. Javier Cobos. Disponible en línea: <http://www.elmundo.es/blogs/elmundo/totalsiessoloapretarunboton/2011/01/28/html5-grandes-mitos-al-descubierto.html>, [acceso Octubre 2011]

[11] New Riders. “Introducing HTML 5”. Bruce Lawson, Remy Sharp

[12] El Futirifoken. “HTML5 : Guardar el contenido de <canvas> en el servidor”. Gazer. Disponible en línea: <http://www.gazer.com.ar/2010/04/13/html5-guardar-el-contenido-de-canvas-en-el-servidor/>, [acceso Noviembre 2011]

[13] Astatic notes | Blog. “Adaptive Thresholding using Integral Image [source]”. Eugene Zatepyakin. Disponible en línea: <http://blog.inspirit.ru/?p=322>, [acceso Octubre 2011]

[14] Adobe Press. "ActionScript® 3.0 for Adobe® Flash® Professional CS5 Classroom in a Book®". Adobe Creative Team.